Fachgebiet Wissensverarbeitung
Institut Künstliche Intelligenz
Fakultät Informatik
der
Technischen Universität Dresden

Diplomarbeit zum Thema:

A resource–oriented deductive approach towards hierarchical planning

von
Kerstin Eder

Mai 1995

Technische Universität Dresden
Fakultät Informatik

Aufgabenstellung für die Diplomarbeit

Name, Vorname des Diplomanden:	Eder, Kerstin
Immatrikulationsnummer:	1180398

Thema: Hierarchisches Planen in der Linearen Logischen Programmierung

Zielstellung:

Die Aufgabe besteht in der Untersuchung von Möglichkeiten für die Lösung hierarchischer Planungsprobleme mit Hilfe eines ressourcen-orientierten deduktiven Ansatzes.

In Analogie zur Wirkungsweise der Chemischen Abstrakten Maschine soll herausgefunden werden, wie in der Linearen Logischen Programmierung die Behandlung von Aktionen als Ressourcen realisiert werden kann. Angestrebt wird, Aktionen so zu gestalten, daß sie generiert und verbraucht werden können. Insbesondere soll dazu die Idee des Membrankonzepts der Chemischen Abstrakten Maschine benutzt werden.

Unter Verwendung mehrerer Beispiele sollen die Möglichkeiten für hierarchisches Planen mit oben genanntem Ansatz systematisch erschlossen werden, so daß in einer nächsten Phase die gefundenen Methoden formal im Rahmen einer Gleichungslogik beschrieben werden können. Ziel ist dabei ein Kalkül zur Behandlung hierarchischer Planungsprobleme. Die entwickelten Methoden sind abschließend in einem Logikprogramm zu implementieren.

Betreuender Hochschullehrer:	Prof. Steffen Hölldobler
Institut:	Künstliche Intelligenz
Fachgebiet:	Wissensverarbeitung
Beginn am:	16. November 1994
Einzureichen am:	15. Mai 1995

Verteiler: Prüfungsamt
HSL
Diplomand

____
Unterschrift des betreuenden Hochschullehrers

Dresden, 29. Mai 1995

Erklärung

Hiermit erkläre ich, daß ich die vorliegende Arbeit selbständig angefertigt und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Kerstin Eder

Abstract

A new concept for the representation of knowledge about dynamic systems in the context of reasoning about actions and change is introduced. It is based on the equational logic programming approach [HS90], which has proven to be an adequate foundation for representing knowledge for planning problems. The basic new idea is the reification of both actions and facts, allowing them to be handled as resources which can be produced and consumed during the planning process. The concept uses a chemical metaphor [BB90] in which situations are represented as solutions in which floating molecules can interact freely according to interaction rules. Situations can be stratified by encapsulating subsituations within membranes such that reactions can occur locally. The membrane mechanism enables us to deal with abstraction and hierarchical planning. Using a simple example, we illustrate step by step how conventional, as well as hierarchical, planning problems can be solved with the new concept. As a concrete deductive approach to reasoning about actions, we show that and how the equational logic programming approach can be extended to the new concepts. We also show that this approach coupled with the new concept can be mapped onto an extended version of the Chemical Abstract Machine. As this machine is a model for parallel processes this may lead to a parallel computational model for reasoning about situations, actions, and causality.

Contents

Chapter 1

Introduction

1.1 Overview

For approximately forty years the artificial intelligence community aims for an operational model of human perception and intellect. The two main aspects of artificial intelligence in which both cognition and engineering take effect, are:

1. cognitive modeling, i.e. simulation of cognitive processes by means of information processing models, and

2. construction of intelligent systems, which incorporate certain human abilities as to perceive and reason.

The ability of an agent to reason about its environment and to modify its behavior to achieve certain goals is definitely one of the most remarkable characteristics of intelligent behavior. In particular this involves to deal with the laws of causality, viz. the connection between cause and effect, its adequate representation, and its efficient use in certain situations. Within this context there arise four categories of problems, each of which groups a set of fundamental questions appearing out of the use of general laws of causality.

The *prediction problem* covers the problem of predicting the effects of a certain behavior (within this context mostly referred to as *actions*). More formally the prediction problem focuses on the search of a description of *the* situation, which is expected to result from the application of a given sequence of actions on an equally given initial situation.

By way of contrast, the *postdiction problem* aims to draw conclusions on the state of the initial situation from one or more observations during or after the application of a sequence of actions. The main characteristics, distinguishing the prediction from the postdiction problem is that in the latter the initial situation is either not at all or only partly specified. However, instead of this information, the effects of the given sequence of actions are provided by means of observations, which makes a further gain of knowledge about the probable initial situation possible.

The two problems we mentioned so far have one common feature, which is that the considered sequence of actions is given right from the beginning. If this is omitted, the prediction problem can be generalized to the more complex problem of *planning*, which today belongs to the more application oriented questions to be solved in the scope of reasoning about causality. Planning deals with the question of generating a sequence of actions which transforms a given initial situation into a so-called goal situation, specified by a particular set of conditions. The last of the four categories of problems can be obtained in the same manner by generalizing the postdiction problem to a problem, where both the initial situation and the sequence of actions are either not at all, or just partly specified, such that from the given observations conclusions need to be drawn on the sequence of actions as well as on the initial situation.

In order to find solutions to the previously introduced problem classes with formal methods, as e.g. logic, one needs to model the reality in a way, such that these methods can be applied. As in every attempt to formalize parts of the reality, there arises the problem of that abstractions of this kind always represent just idealizations of the real world [Sch77]. Consequently, the formal answer to a problem might not solve the corresponding real world problem, since through the formalization we have lost some of the details that were relevant to solve the problem properly.

In the following we are going to list some of the difficulties generally known to complicate the formalization and automation within the field of reasoning about causality. The first of them is the *qualification problem*, which addresses the fact, that it is impossible to state *all* the conditions necessary to execute an action successfully. A common example often mentioned in the literature [Gin93] in this connection is the attempt to specify the preconditions for starting a car. One certainly includes such conditions like the key is in the ignition and there is gas in the tank. Typically, however, one would not take into account the condition that there is no banana in the tail pipe, although this situation could conceivably arise and prevent the car from starting[1]. This example suggests that we may describe simplified scenarios, in which we need only mention conditions being actively considered.

On the other hand *all* the effects of a successfully performed action can — in general — not be specified; this is commonly described as the *ramification problem*. As an example, imagine a tray with a bowl of breakfast cereal standing on it. By moving the tray from the kitchen to the dining room, the bowl too changes its location.

The third of these problems is the *frame problem*, which deals with precisely describing what stays the same when an action does succeed. For illustration, imagine to have five quarters in your purse and there was a machine which sells a tea for three quarters. A common assumption in planning is, that unless an action explicitly causes a certain fact to hold or not to hold, the facts are preserved by the action. In the example, if you spent three quarters by buying some tea, it is only natural that you expect the remaining two quarters still to be in your purse.

Tackling all these problems in a natural way that promotes efficient reasoning is a major challenge in planning. An efficient planner either solves all the problems efficiently or avoids them by restricting itself to certain assumptions, as all the mentioned problems are in general undecidable [AKPT91]. Consequently, because of the existence of the addressed problems, the soundness and completeness of planning systems is limited with respect to their application area. From the reflections we made so far, it becomes clear, that the formalization of real world states, which is characteristic for solving planning problems with formal methods, results in a very high complexity within knowledge representation for planning.

[1] As it occurred to Eddie Murphy in the 1985 film "Beverly Hills Coup".

The first attempt towards modeling action in artificial intelligence has been to view action as state change. This view is the foundation for all the state-based planning systems (like STRIPS [FHN72, Lif86] and its successors), and early formal models of planning (like the situation calculus [McC63, MH69]). It models the world by a succession of states (or situations), each state representing an instantaneous *snapshot* of the world formally described by propositions which are relative to the states. Actions are defined as functions from one state of the world to another.

In particular many theories of actions (e.g. [Rei91, GL93, San94]) have been based on the ideas underlying the situation calculus. There a situation is a snapshot of a particular part of the universe at a specific point of time, represented by a set of logical formulas. Facts of a situation are represented by *fluents*, which are functions (or predicates) whose domain contains the space of situations. If the range of a fluent is the set of truth values then a fluent is called *propositional fluent*. If the range is the space of situations then a fluent is called *situational fluent*. An action is a formal description of a real world action. It transforms one situation into another. The situation calculus is a methodology for expressing facts about action and change in formal languages of mathematical logic. It states how from a given situation $\mathcal{S}$ and an action, the successor situation $\mathcal{S}'$ can be obtained.

In every formal model which understands situations as parts of the real world at a particular point of time the dynamic aspect of the changing situations, i.e. the transition between two situations needs to be considered. When describing a situation in classical logic a formula is either true or false, which is in our context inadequate since the truth-value of a proposition is time-dependent, i.e. it might change with time. In order to substitute this lack of expressiveness, in the situation calculus each fluent and all the action descriptions were augmented by an additional argument, representing the situation in which they are valid.

Although the situation calculus seems to have sufficient expressive power, it has deficiencies from a computational point of view. For example, in order to overcome the frame problem it is necessary to add axioms, the so-called frame axioms, expressing which values of fluents are not changed when actions are executed. However, since we need $O(m \times n)$ axioms, where m is the number of fluents and n is the number of actions, using frame axioms is not a very good idea. Moreover the axioms may lead to many redundant derivations, slowing the whole computation process down. Even if the number of frame axioms is reduced by choosing a better representation as done in e.g. [Rob67, Kow79], the solution remains unsatisfactory. McCarthy then proposed to use non-monotonic inference rules like circumscription [McC90, McC86], which has led to numerous further publications and problems.

The deficiencies, which show up in the need to state frame axioms or a non-monotonic law of inertia, can ultimately be traced back to the representation of the facts (or fluents) describing a situation [GHS$^+$92, HT95b]. The additional argument of a fluent, representing the situation in which the fluent holds, needs to be updated each time an action is performed even in case the fluent is not affected by the action. From observations like this it becomes clear that the method of knowledge representation underlying a particular formalism determines the strength and expressiveness of the whole formalism, and consequently new ways of knowledge representation for reasoning about situations, actions, and causality need to be investigated. In the next section we will informally introduce one of these new approaches giving the motivation for the research undertaken within this thesis.

1.2 Motivation

In recent years new deductive approaches have been developed which enable us to model situations, actions, and causality without the need to state frame axioms or non-monotonic laws of inertia [Bib86, MTV90, HS90]. Instead of representing the atomic facts of a situation as fluents, these approaches represent the facts as resources. Resources do not hold forever — as fluents do — but are consumed and produced by actions. Consequently, resources which are not affected by an action remain as they are — they need not be updated.

In particular, the equational logic programming approach (ELP, for short), developed in [HS90], has some nice properties. It is a first–order formulation, admits a standard semantics, and has been shown to be equivalent to the other resource–oriented approaches [GHS95]. It has also been extended to handle objects [GHS$^+$92], specificity [HT95a], concurrent [BT94] as well as non-deterministic actions [BHS$^+$93, Thi94c] and ramifications [Thi95a]. Moreover, its relation to the high-level specification language $\mathcal{A}$ [GL93] and various of its dialects, as well as to the Ego–World–Semantics [San94] has been clarified [Thi94c, Thi94a]. The major aim of this work is to extend the equational logic programming approach even further to handle hierarchical plans, but the most crucial innovation is a new view on the notion of an action.

In all — conventional as well as resource–oriented — approaches to dynamic systems, actions have been represented as partial descriptions of functions which, if applied to a situation, transform this situation into a subsequent situation, where an action is only applicable if all its conditions hold (or are available) in the current situation. Thereby, it is assumed that actions are always available for application. A closer examination of this paradigm to describe the world, however, will show that, in reality, there are situations where the availability of an action is not necessarily guaranteed at each instant of time. Similarly it is possible to just partly fulfill the conditions of an action, thus achieving an opportunity to continue the same action eventually. Once all the conditions are fulfilled, the action is activated and its effects can be obtained.

Let us look at an example adapted from [GHS$^+$92]. Suppose a thirsty person named Bert wants to get some tea from a vending machine. The tea costs 75 cents, which should not be a problem as Bert has a one dollar note and a quarter in his jacket. Unfortunately, the vending machine accepts only quarters. But there is also a change machine which will give Bert four quarters for a dollar note. The problem of getting some tea can then be described as a planning problem with the initial situation of Bert having a dollar note (d) and a quarter (q), the operators *get–change* and *get–tea* which allow him to get change and to get some tea, respectively, and the goal where Bert has a cup of hot tea (t) in his hand. An obvious solution to this problem is the plan with the two consecutive actions *get–change* and *get–tea*.

Now, since Bert initially has got only one quarter, he might insert this one, realize that he needs two more, go and change the dollar note, and then insert the remaining two quarters. Situations like the one reached after having inserted the first quarter cannot be expressed within conventional approaches as these only allow an action to be applied if all the conditions are fulfilled (i.e. are available at the same time). In this work, we propose to describe this particular situation by introducing a new *get–tea* action as an opportunity to get some tea for only two quarters. This way we solve the planning problem by starting the *get–tea* action, stopping in order to do the *get–change* action, and finally continue and finish the *get–tea* action to obtain some tea.

The second major contribution of this work deals with the consideration of different levels of abstraction, in view of hierarchical planning. In the example, the condition for getting some tea is the possession of three quarters, and Bert expects the machine to provide some tea after he inserted the coins. But, what if there is a power failure, no water supply or the machine is out of tea bags? Obviously we have described the action on a relatively high level of abstraction where we assumed that electricity, water, and tea bags are always available. Described on a lower level, the same action might have more conditions, like electricity (e), water (w), tea bags (b) plus, as before, three quarters. Let us assume that the machine has got a permanent supply of electricity and water such that we do not have to consider them in greater detail. The tea bags, however, need to be restocked regularly. The maximum capacity is at, say, 100 tea bags. Therefore, we assume that after adding new tea bags, the resources electricity, water and 100 tea bags are available such that, on a higher level of abstraction, there are 100 opportunities to get some tea for "just" three quarters. Furthermore, a new more complex machine could be created by combining the actions *get–change* and *get–tea* into one new action with a dollar note as condition and a quarter plus a cup of tea as effects. Again, this more abstract action is a resource, as such a modern vending machine has only a finite number of tea bags and change.

In the following we want to distinguish between basic actions, which are available without investing anything (i.e. those on the lowest level of the current abstraction which are specified by all the conditions needed on this abstraction level), and what we called opportunities above, which are in fact obtained from basic actions by adding those resources from which we want to abstract on the next higher level (e.g., like the abstraction from needing electricity, water and tea bags to get some tea). The latter can be seen as a type of action where something has already been invested to achieve their effects, hence they are not endlessly available. In other words, an opportunity can be produced either by adding certain resources to the corresponding basic action or to an opportunity on the next lower level, or else by combining basic actions or opportunities. By doing this, these opportunities enter a new state, similar to some product passing through different production processes and entering a new state each time until the final version is assembled. An opportunity therefore represents something which contains resources and, for this reason, it is consumed when all its conditions are fulfilled and the effects are obtained. In short, actions and opportunities should also be treated as resources!

Now, the question is how basic actions and opportunities can be represented in a deductive system. In ELP resources are essentially represented by multisets of terms. Can we reify actions and opportunities as well? If we do so, then obviously we can no longer use the conventional view on planning, viz. that only actions transform one situation into another. Rather, as the available actions are now an element of the situation, the various interaction rules between actions and facts as well as between actions define the relationship between situations.

The new view on planning is based on a chemical metaphor where a situation is represented as a *chemical solution* in which floating *molecules* can interact according to *interaction rules*. The idea is adapted from a paper by G. Berry and G. Boudol [BB90] in which the *Chemical Abstract Machine* (or CHAM, for short) is introduced as an abstract machine for modeling asynchronous concurrent computations. There, a solution is represented by a multiset of molecules, hence the interaction rules are essentially multiset rewritings. To deal with abstraction and hierarchies, a molecule may contain a subsolution enclosed in a membrane. Such membranes can be somewhat porous to allow communication between the encapsulated solution and its environment. Basically the CHAM has no control mechanism, and the whole interaction process is non-deterministic. It is assumed that the solution is stirred automatically, thus contact between any two molecules is possible. There are different interaction rules, some of which result in structural changes only, while others, the so-called *reactions*, consume molecules and produce new ones.

Applying this to the world of planning problems, we will represent a situation as a solution containing facts and actions. Note, that this means representing both facts and actions in the same way. The elements floating in a solution can interact freely according to interaction rules. During a reaction two of these elements are consumed and a new one is produced. A planning problem is then formalized by a solution modeling the initial situation, another solution representing the goal situation, and the question of whether there exists a sequence of interactions (determining a *plan*) transforming the initial situation into a solution satisfying the goal specification. For instance, the initial situation of the previously described tea scenario can be considered as a solution containing four elements representing the facts *dollar note* and *quarter* plus the actions *get–change* and *get–tea*. The goal is then a solution containing, amongst others, an element representing the desired *tea*.

The strength of this concept lies in the reification of both actions and facts, enabling us to handle actions as resources which is essential for modeling hierarchical planning. With the new membrane notion it is possible to stratify a situation by encapsulating subsituations within membranes where interactions can occur locally. Thus, it becomes possible to produce new complex actions and to provide these for reactions in their environment.

A new question arising out of the just proposed representation of actions as resources within a chemical metaphor is whether such a deductive, hierarchical planning system can be implemented efficiently. Running a system like this on a standard PROLOG-system does not seem to be the most promising answer. Within this work we therefore examine a way of mapping the extended ELP–approach onto a slightly modified version of the CHAM, called PCHAM. As this machine is a model for parallel processes this may lead to a parallel computational model for reasoning about situations, actions and causality.

On the whole, employing the just proposed new view on actions as resources seems to be a very promising extension to ELP, especially with regard to modeling abstraction and hierarchical planning problems. Within this thesis we therefore want to do a first step in this direction aiming to achieve a model for knowledge representation and reasoning about situations, actions, and causality which incorporates this new view on actions and demonstrates how it can actually be employed to solve conventional as well as hierarchical planning problems.

1.3 Outline

The rest of the thesis is organized as follows. Chapter 2 contains the basic concepts, vocabulary and definitions, viz. a short introduction to equational logic programming and multisets, and a brief review on some earlier work on resource-oriented deductive approaches towards planning. Furthermore, in the last section of Chapter 2 we formally introduce the abstract machine model of the CHAM, and give some examples. Chapter 3 is the main chapter concerning the extension of the resource-oriented knowledge representation concept for ELP. In Section 3.1 we formally show how situations and actions can be represented as resources. We start by demonstrating that the new approach captures the previous work on equational logic programming and planning (viz. situations and basic actions). Thereafter, we gradually extend this work by allowing more complex opportunities than basic actions. We also introduce generators which, similarly to exponentials in linear logic [Gir87], allow duplication of actions in a controlled manner. In Section 3.2 we then demonstrate how hierarchical planning can be modeled in the new approach in a natural way and without the need of frame axioms or other non-monotonic laws of inertia. We extend the equational logic program in order to compute in hierarchical domains. Section 3.3 contains a short discussion of alternative methods for extracting a plan from a deductive planning system. In Chapter 4 we introduce an abstract machine model for hierarchical planning. Section 4.1 contains some preliminary thoughts about reasons for modifying the CHAM to suit planning purposes. In Section 4.2 we introduce a new example scenario which will be used when demonstrating the functionality of the modified CHAM which is described in Section 4.3. Finally, in Chapter 5 we conclude and give an outlook on further research.

Parts of this work have been submitted to the International Logic Programming Symposium 1995 [EHT95].

Acknowledgements

I am particularly grateful to my supervisor Prof. Steffen Hölldobler for his constant support, crucial guidance, advice and encouragement during this work. He greatly influenced my attitude to research and teaching, and I enjoyed working with him.

I am also thankful to Michael Thielscher and Sven-Erik Bornscheuer for the fruitful discussions which helped to straighten out the ideas underlying the new concept introduced within this work.

Special thanks are due to Eckhard Koch, Katalin Prasser, and Christian Urban for proof-reading and commenting on an earlier version of this thesis.

Finally, I want to thank my parents Helga and Hans-Georg Eder for their constant support and appreciation throughout the years. For inspiration, encouragement and other reasons I would also like to thank Kathrin and her family, Innette, Vida and my sister Antje.

Chapter 2

Foundations

This chapter provides some basic definitions and a brief review of notions and notations concerning logic programming and unification under an equational theory which forms an essential foundation for this work. As multisets play an important role for the semantics of situations and actions, this chapter contains a formal introduction to the concept of multisets. Furthermore a review on some previous work on approaches towards reasoning about situations, actions, and causality is given, followed by a short overview on the main features of the Chemical Abstract Machine.

2.1 Foundations of Equational Logic Programming

An important part of this work is to extend the knowledge representation concept used in ELP. As ELP is based on logic programming, we first want to give a short introduction on the foundations of logic programming and equational logic programming by stating some basic definitions. The reader is assumed to have a basic understanding of the vocabulary and the standard theoretical results of logic and logic programming, which are contained in [Men79, Llo87, Höl89].

Definition

A *term* is defined inductively as follows: A variable is a term, a constant is a term, and if f is an n-ary function symbol ($n \geq 1$) and $t_1, \ldots, t_n$ are terms, then $f(t_1, \ldots, t_n)$ is a term. A *ground* term is a term not containing variables.

An *atom* is an expression of the form $p(t_1, \ldots, t_n)$, where $t_1, \ldots, t_n$ are terms, and p is an n-ary ($n \geq 0$) predicate symbol. A *ground* atom is an atom not containing any variables.

A *literal* is either an atom or a negated atom.

A *normal E-program* is a pair (P, E) where E is an equational theory and P is a finite set of program clauses of the form $A \leftarrow L_1 \wedge \ldots \wedge L_n$, where A is an atom and $L_1, \ldots, L_n$ $(n \geq 0)$ are literals. The atom A is called the *head*, and the literals $L_1, \ldots, L_n$ form the *body* of the clause. Any free variables occurring in either the head or the body are assumed to be universally quantified in front of the respective clause. If the body of the clause is empty ($n = 0$), the clause is called *unit clause* or *fact*.

Definition
An *equational theory* E is a set of universally quantified expressions of the form $s = t$, where s and t are terms. Two terms are called *E-equivalent* (written $s =_E t$), if they are equal with respect to E. An equational theory is called *regular* iff for all its equations $\forall X_1, \ldots, X_n . s = t$ we have

$$\mathcal{V}ar(s) = \mathcal{V}ar(t) = \{X_1, \ldots, X_n\}^1.$$

Definition
A *normal goal* is a clause of the form $\leftarrow L_1 \wedge \ldots \wedge L_n$, where $L_1, \ldots, L_n$ $(n \geq 0)$ are literals. Any free variables are assumed to be universally quantified at the front of the goal. If $n = 0$ then the so called *empty goal* is denoted by $\Box$.

Definition
A *substitution* θ is a finite set of the form $\{v_1 \mapsto t_1, \ldots, v_n \mapsto t_n\}$, where each v_i is a variable, each t_i is a term distinct from v_i and the variables $v_1, \ldots, v_n$ are distinct. The substitution given by the empty set is called *identity* or *empty substitution*; it is denoted by ε.

The *application* of a substitution θ on a term t is defined by $t\theta = \theta(t)$ if t is a variable and $t\theta = f(t_1\theta, \ldots t_n\theta)$ if $t = f(t_1, \ldots, t_n)$. The *restriction* of a substitution θ to a set of variables $\mathcal{V}$, written $\theta|_\mathcal{V}$ is defined as

$$\theta|_\mathcal{V} = \{\ x \mapsto t \in \theta \mid x \in \mathcal{V}\ \}.$$

Let σ and θ be substitutions, the substitution $\sigma\theta$ is called the *composition* of σ and θ, and is defined by $X(\sigma\theta) = (X\sigma)\theta$ for all variables X.

Within this thesis substitutions will be denoted by Greek lowercase letters such as γ, λ, σ and θ.

Definition
Let E be an equational theory. An *E-unification problem* for two terms s and t is the question of whether there are substitutions θ with $\mathcal{V}ar(\theta)^2 \subseteq \mathcal{V}ar(s) \cup \mathcal{V}ar(t)$ such that

$$s\theta \ =_E \ t\theta.$$

A substitution solving the above E-unification problem is called *E-unifier* for s and t. Two terms are called *E-unifiable* iff they have at least one E-unifier.

An *answer substitution* θ for a normal goal $\leftarrow G$ wrt. a normal E-program (P, E) is the substitution obtained by restricting the composition $\theta_1 \ldots \theta_n$ to the variables of G, where $\theta_1 \ldots \theta_n$ is the sequence of E-unifiers used in an SLDENF-refutation of length n of $P \cup \{\leftarrow G\}$.

2.2 Using Multisets to Represent Resources

2.2.1 Multisets

Obviously the most important property distinguishing resource-oriented approaches from conventional approaches to planning is the use of resources which can be produced and consumed. In order to model

[1] Where $\mathcal{V}ar(t)$ denotes the set of variables contained in the term t.
[2] Where $\mathcal{V}ar(\theta)$ denotes the set of variables contained in the substitution θ.

resources adequately, the concept of multisets has been introduced. Like in previous publications on ELP [HS90, HT93, Thi94c] we also want to denote situations by multisets of resources where consumption means removing elements and production means adding elements.

The crucial difference between ordinary sets and multisets it that a multiset might contain an element more than once. For example adding the element b to the set $\{a, b, c\}$ would not make a difference, but adding the same element to the multiset $\{\!\!\{ a, b, c \}\!\!\}$ yields the multiset $\{\!\!\{ a, b, c, b \}\!\!\}$. Hence two multisets are equal if and only if each element contained in one of them with a certain number, occurs equally often in the other.

Definition
A *multiset* is a set-like collection of elements, in which by contrast to classical sets elements may occur more than once. For denoting multisets we use the brackets $\{\!\!\{$ and $\}\!\!\}$ and the operators $\dot{\cap}$, $\dot{\cup}$, $\dot{-}$, and $\dot{\subseteq}$ to denote the multiset operations corresponding to the set operations $\cap$, $\cup$, $-$, and $\subseteq$, respectively.

In the multiset $\mathcal{M} = \{\!\!\{ a, b, c, b \}\!\!\}$ the element b occurs twice, which can be written as $b \in_2 \mathcal{M}$, the element d does not occur in $\mathcal{M}$, which is written as $d \in_0 \mathcal{M}$ (or equivalent $d \notin \mathcal{M}$). The union $\mathcal{M} \dot{\cup} \mathcal{N}$, the intersection $\mathcal{M} \dot{\cap} \mathcal{N}$, the difference $\mathcal{M} \dot{-} \mathcal{N}$ and the subset relation $\mathcal{M} \dot{\subseteq} \mathcal{N}$ between two multisets are defined as follows:

$$\begin{array}{rcl}
e \in_m \mathcal{M} \wedge e \in_n \mathcal{N} & \rightarrow & e \in_{m+n} \mathcal{M} \dot{\cup} \mathcal{N} \\
e \in_m \mathcal{M} \wedge e \in_n \mathcal{N} & \rightarrow & e \in_{min(m,n)} \mathcal{M} \dot{\cap} \mathcal{N} \\
e \in_m \mathcal{M} \wedge e \in_n \mathcal{N} \wedge m > n & \rightarrow & e \in_{m-n} \mathcal{M} \dot{-} \mathcal{N} \\
e \in_m \mathcal{M} \wedge e \in_n \mathcal{N} \wedge m \leq n & \rightarrow & e \in_0 \mathcal{M} \dot{-} \mathcal{N} \\
\mathcal{M} \dot{\subseteq} \mathcal{N} & \leftrightarrow & \forall e.[e \in_m \mathcal{M} \rightarrow \exists n.[e \in_n \mathcal{N} \wedge n \geq m]]
\end{array}$$

Furthermore is $\mathcal{M} \doteq \mathcal{N}$ iff $\mathcal{M} \dot{\subseteq} \mathcal{N}$ and $\mathcal{N} \dot{\subseteq} \mathcal{M}$.

2.2.2 Predicates over Multisets

For representing multisets as terms in logical programs we use the binary function symbol $\circ$. It is assumed to be associative (A), commutative (C), and admits a unit element (1) denoted by $\emptyset$. Informally the data structure multiset can be represented by connecting the elements contained in a multiset via the new function symbol; the resulting term is then called an AC1-term.

In order to transform the equality relation $\doteq$ between multisets to its adequate representation when acting on AC1-terms, we need the following equational axioms, which together with the axioms of equality, i.e. reflexivity, symmetry, transitivity, and substitutivity form the equational theory AC1[3]:

$$\begin{array}{rcll}
(X \circ Y) \circ Z & = & X \circ (Y \circ Z) & \text{(A)} \\
X \circ Y & = & X \circ Y & \text{(C)} \\
X \circ \emptyset & = & X & \text{(1)}
\end{array}$$

In [Thi94b] a function symbol φ has been introduced, which formally transforms the multisets into their equivalent representation in equational logic programs; note that the constant $\emptyset$ represents the

[3] In the following all variables are assumed to be universally quantified.

empty multiset. The definition of φ is given inductively by the following equations. Let X be a variable, c a constant, f a function symbol, e a unary function symbol denoting that its argument is enclosed in multiset brackets, and $\mathcal{A}_1, \ldots, \mathcal{A}_n$ arbitrary multisets.

$$\begin{aligned}
\varphi(X) &= X \\
\varphi(c) &= c \\
\varphi(\{\!\!\{\ \}\!\!\}) &= \emptyset \\
\varphi(\{\!\!\{\ \mathcal{A}_1, \ldots, \mathcal{A}_n\ \}\!\!\}) &= e(\varphi\mathcal{A}_1) \circ \ldots \circ e(\varphi\mathcal{A}_n) \\
\varphi(\mathcal{A}_1 \dot{\cup} \ldots \dot{\cup} \mathcal{A}_n) &= \varphi\mathcal{A}_1 \circ \ldots \circ \varphi\mathcal{A}_n \\
\varphi(f(\mathcal{A}_1, \ldots, \mathcal{A}_n)) &= f(\varphi\mathcal{A}_1, \ldots, \varphi\mathcal{A}_n)
\end{aligned} \tag{2.1}$$

The predicates used in ELP are predicates over the data structure multiset. This enables us to model the transformation of situations as a process of consumption and production of resources, which in fact simulates the processes happening in real world domains. In the original ELP–approach [HS90] situations are multisets of facts, and a fact itself is a simple term — simple in that it is not a multiset itself. Consequently terms of the form $\emptyset$ or $t_1 \circ \ldots \circ t_n$, with $n \geq 1$, are used to represent situations, where the t_i 's denote facts which are simple terms not containing the function symbol $\circ$. As proposed in [Höl92] if such terms are ground they can be mapped to multisets with the help of a function ψ as follows.

$$\begin{aligned}
\psi(\emptyset) &= \{\!\!\}\!\!\}, \\
\psi(t_1 \circ \ldots \circ t_n) &= \{\!\!\{t_1\}\!\!\} \dot{\cup} \psi(t_2 \circ \ldots \circ t_n).
\end{aligned} \tag{2.2}$$

The inverse function to ψ is a mapping from ground multisets into AC1–terms. It is denoted by ψ^{-1}, and is defined as follows.

$$\begin{aligned}
\psi^{-1}(\{\!\!\}\!\!\}) &= \emptyset, \\
\psi^{-1}(\{\!\!\{t_1, \ldots, t_n\}\!\!\}) &= t_1 \circ \psi^{-1}\{\!\!\{t_2, \ldots, t_n\}\!\!\}.
\end{aligned} \tag{2.3}$$

One should observe that $\psi^{-1}(\psi(s)) =_{AC1} s$ and $\psi(\psi^{-1}(\mathcal{S})) \doteq \mathcal{S}$, where s is an AC1–term and $\mathcal{S}$ is a multiset denoting a situation. Furthermore, $\psi(s) \doteq \mathcal{S}$ iff $s =_{AC1} \psi^{-1}(\mathcal{S})$. We will define similar mappings for the new approach in Section 3.2.4.

2.3 Review on Previous Work

In this section we are going to pursue the development of several methods towards reasoning about situations, actions, and causality. We want to start with further examining McCarthy's work on this topic the basic ideas of which have been presented in the introduction, followed by a short presentation of the foundations underlying STRIPS, and a discussion of both methods. We then focus on resource-oriented deductive approaches, present some of the various extensions made to the equational logic programming approach, and finally give an idea of how the work presented within this thesis fits into the extensions of ELP. In Section 2.4 we will give a deeper insight into one of the resource-oriented approaches, viz. the equational logic programming approach.

2.3.1 First Steps Towards Reasoning about Situations, Actions, and Causality

The first proposal for formalizing reasoning about situations, actions, and causality is due to McCarthy and Hayes [McC63, MH69]. It suggests to state explicit axioms for inferring what fluents hold after each possible action. There are two sorts of axioms, some of which indicate the things that change when each action takes place (called action axioms), others indicate the things that remain unaffected by the application of an action (called frame axioms). The problems arising from this approach are on one hand epistemological, on the other hand they are of computational nature. The epistemological difficulty is that of providing explicit frame axioms for each action, i.e. stating under what circumstances facts involving the action do not change when the action is performed. In general $O(m \times n)$ frame axioms are required, where m is the number of fluents and n is the number of actions, in order to determine which fluents persist from one time to the next. The computational difficulty arises out of the fact that when there are many fluents in the world model, to determine what is true after the application of some action, the reasoner must examine every fluent, and prove whether it still holds in the successive situation. In short, the approach is cumbersome because the set of frame axioms required is large and complex, and causes the reasoner to have very poor performance.

The main problem of McCarthy's approach is that one needs to state all the things that do not change explicitly. One would rather like to provide axioms only for the things that change, and introduce a single default axiom stating that everything else persists. In order to do this a non-monotonic formalism [McC90, McC86] needs to be employed, which allows us to write an axiom stating that facts persist in the absence of information to the contrary. By using this method, the number of frame axioms can be reduced. However, the approach still suffers from computational problems since the default frame axiom needs to be examined once for every fact in order to determine what is true after the application of an action.

Another approach to reasoning about situations, actions, and causality is the STRIPS approach [FHN72]. Its basic idea is that the universe does not change much from one state to the next. Hence in STRIPS a single model of the world is kept, and updated to represent the changes resulting out of the performance of an action. Situations are represented by sets of single positive fluents which are known to be true in the corresponding real world situation. In addition the negation of fluents not contained within this set is implicitly assumed. In STRIPS, an action is described by means of its preconditions, and two sets, called add list and delete list. An action can be performed if all its preconditions are satisfied. The result of the execution is to add the facts contained in the add list to the current world model, and to delete the facts contained in the delete list from it. In fact, since situations are represented by sets, the successor situation $\mathcal{S}'$ of a situation $\mathcal{S}$ can be obtained by set difference of $\mathcal{S}$ and the set of fluents contained in the delete list, and subsequent set union of the rest of $\mathcal{S}$ and the set of fluents in the add list. In this manner the STRIPS approach provides an effective and elegant solution to the frame problem. No additional axioms are needed, and fluents not contained in neither the add nor delete list, i.e. fluents which are not affected by the performance of an action continue to hold automatically in the successor situation, because the set operations difference and union do not have an effect on them. However, since all conditions, and all consequences of actions must be listed explicitly either in the preconditions, or in the add and delete lists, i.e. it is assumed that an action is always specified completely and correctly (which is commonly known as the STRIPS assumption), this approach fails to handle inferred consequences, and thus does not solve the ramification problem, nor it allows for possible solutions to the qualification problem. Moreover, the user of the system is responsible for maintaining appropriate descriptions of the actions, and as the problem domain becomes bigger this becomes a very complicated task. On the other hand the use of sets of atomic formulas and the performance of operations on them escapes the scope of pure first order logic, and thus finding a satisfactory semantics for the STRIPS approach took a big effort and a long time [Lif86]. Furthermore, the result found placed several restrictions on STRIPS, one of which is the requirement that the initial situation always needs to be specified completely, hence when thinking about the different problems discussed in the introduction, the STRIPS approach is not suited for solving postdiction problems.

As it can be seen the first steps towards a formalization of reasoning about situations, actions, and causality did not solve the addressed problem satisfactory. However, they form attempts which disclosed ways for possible solutions, gave raise to problems which would not have been considered before, and thus are valuable contributions on the way towards finding new and better formalisms.

2.3.2 The Resource-Oriented Approaches

For solving planning problems declaratively and without the need to state extra frame axioms three deductive methods have been developed in recent years. In the following we briefly want to introduce the basic ideas underlying them with the help of the tea scenario described in the introduction. In [GHS95] it is shown that these approaches are equivalent, and that they are not only suited to handle deductive planning problems, but can be employed whenever actions are to be modeled in logic.

One of the deductive approaches to planning is based on the connection method, it is called *linear connection method*, and was introduced by W. Bibel [Bib86]. In the linear connection method the usual inference rules of the connection method are restricted such that in a proof each literal is engaged in at most one connection. Hence, the connection of a literal during the inference process is like the consumption of the fact (or resource) that is represented by this literal. In the opposite way, the satisfaction of the conditions of an implication allows to use the literals contained in the conclusion for further connections, which can be regarded as the production of these literals.

The initial situation of our example can be represented by the following axiom

$$\exists Z.\ [s(Z) \land d \land q],$$

and the actions get–change (g_c) and get–tea (g_t) are denoted by the axioms

$$\forall S.\ [s(g_c(S)) \land d \to q \land q \land q \land q \land s(S)],$$
$$\forall S.\ [s(g_t(S)) \land q \land q \land q \to t \land s(S)].$$

The goal is a formula of the form $s(\Lambda) \land t$, where the constant Λ denotes the empty plan. In order to record the actions performed to achieve the goal, the predicate s , a so called *state literal* is used. We do not want to depict the connection proof here. A proof similar to our example is contained in [GHS95]. Nevertheless, we want to state its result, which is the answer substitution $\{Z \mapsto g_c(g_t(\Lambda))\}$, meaning that the get–change action is performed previous to the get–tea action.

The second of the deductive approaches to reasoning about situations, actions, and causality was presented by M. Massaron etal. [MTV90], who proposed the use of the multiplicative fragment of linear logic for solving planning problems. Linear logic, which was first introduced by J.-Y. Girard [Gir87] is a refinement of classical logic. One of the remarkable features of linear logic is its resource sensitiveness as it provides a natural accounting of resources. A proof in linear logic looks very similar to proofs in the Gentzen–style sequent calculus of classical logic, but excludes the use of weakening and contraction rules, thus literals and formulas cannot be copied or erased. In this approach our example can be formalized by the current state axioms $\vdash d$ and $\vdash q$ and by the transition axioms[4]

$$d \vdash q \otimes q \otimes q \otimes q,$$
$$q, q, q \vdash t.$$

The plan solving the problem can be extracted from the proof by means of recording the transition axioms and the order in which they were used in the proof (see [GHS$^+$92]).

The third method — ELP — on which we want to have a closer look at was proposed by S. Hölldobler and J. Schneeberger in [HS90]. It combines the STRIPS–idea with the method of reification of the terms representing situations, and uses classical logic along with an equational theory as a planning formalism. The concept of *reification* enables us to treat logical propositions formally as terms and thus opens up opportunities for formalizing the basic ideas underlying STRIPS in first order logic. Through reification a complete STRIPS–like situation can be represented as a single term, viz. as a multiset of reified fluents (or facts). It can then be represented by connecting its elements via the $\circ$ function symbol, i.e. by an AC1–term.

The planning process in ELP is specified using a ternary predicate $plan(s, p, s')$ which is interpreted declaratively as the execution of plan p — which is a series of instances of actions — transforms situation s into situation s'. For each action there is one triple $\langle \mathcal{C}, a, \mathcal{E} \rangle$, where $\mathcal{C}$ and $\mathcal{E}$ are multisets of facts called the conditions and effects of the action respectively, and a is an identifier for the action. In the equational logic program actions are defined by clauses[5] of the form:

$$plan(\psi^{-1}(\mathcal{C}) \circ V, a(P), W) \leftarrow plan(\psi^{-1}(\mathcal{E}) \circ V, P, W). \tag{2.4}$$

where ψ^{-1} is a mapping from ground multisets into AC1–terms as defined in Subsection 2.2.2, a corresponds directly to the identifier a in the action description, and P denotes a plan. Such a clause is applicable if the conditions are part of the current situation and the remaining facts are bound to the variable V via a specific AC1–unification procedure. In order to terminate derivations the fact

$$plan(V \circ W, \Lambda, V). \tag{2.5}$$

is used, it states that if the current situation establishes the goal situation nothing needs to be done.

Let us illustrate the use of the above clauses with the help of the tea example. The actions get–change and get–tea are specified by the following instances of the clause in 2.4:

$$\begin{aligned} plan(d \circ V, g_c(P), W) &\leftarrow plan(q \circ q \circ q \circ q \circ V, P, W). \\ plan(q \circ q \circ q \circ V, g_t(P), W) &\leftarrow plan(t \circ V, P, W). \end{aligned}$$

The goal $\leftarrow plan(d \circ q, P, t)$ can be resolved with the two clauses above and the clause in (2.5) using SLDE-resolution as the rule of inference, i.e. SLD–resolution augmented by a special purpose unification procedure [Höl89], yielding the answer substitution $\{P \mapsto g_c(g_t(\Lambda))\}$, which corresponds to a plan containing the consecutive performance of the actions get–change and get–tea. The computation involved resembles a computation in STRIPS, except that multisets are used instead of sets and multiset

[4] Where $\otimes$ denotes *multiplicative* conjunction, i.e. the conjunction $d \otimes q$ stands for the proposition that one has both d and q at the same time.

[5] Throughout this work, we use a PROLOG-like syntax, i.e. constants and predicates are in lower cases whereas variables are denoted by upper case letters. Moreover, the free variables are assumed to be universally quantified.

operations are applied instead of set operations. The advantage of the ELP–approach becomes obvious when comparing it with STRIPS which is a non-deductive planning system. ELP is based on a pure first-order logic calculus, and thus admits a standard semantics.

To sum it up, the three deductive methods introduced in this section have one common feature, they all treat facts describing situations as resources which can be consumed and produced when an action is performed, this is why they are called *resource–oriented* approaches. In the following we want to concentrate on several extensions of the ELP–approach to give the reader an idea of the expressiveness and the suitability of ELP for solving problems concerning situations, actions, and causality.

2.3.3 Actions and Specificity

An important limitation on the set of actions defined for a given scenario is the restriction, that for one action there must not be more than one action description. For the formalization of simple scenarios, like our tea example this is no problem. However, it is easy to find other examples, just as simple, in which we face a problem when trying to formalize them, since occasionally there might be different effects resulting from the very same action, depending on the situation it is applied to.

For example imagine a scenario — commonly called the *fragile object domain* [Thi94b] — in which a robot acts on a number of objects, some of which are robust whereas others are fragile. There could be an action named *drop(X)*, defining how a situation changes after dropping an object. In general a condition to apply this action is that the robot holds the object. Obviously, after dropping it, this does not hold any more, and therefore vanishes from the succeeding situation. However, a difference should be made between fragile and robust objects, since certainly after dropping a fragile object it is in addition broken, whereas dropping does not affect a robust object.

If there was the possibility to specify both these actions by means of an additional condition, it became possible to obtain a correct successor situation for both the fragile and the robust object. This works fine as long as the conditions of the actions do exclude one another, which is the case in the above example, as an object can not be fragile and robust at the same time. However, if the robot is holding an object for which there is no information on whether it is fragile or robust, we still want to be able to conclude, that after dropping it, it is not held any more. This could be obtained by introducing a third variant of the action *drop(X)*, which generalizes the previous two variants in that it just requires an object to be held by the robot regardless whether it is fragile or robust.

Dropping a robust object still results in a correct successor situation. But if the robot now drops a fragile object, the resulting situation might not be correct, since there are two applicable actions; and if the one we introduced last is employed, the object remains fragile even though it is certainly broken in reality. *Specificity* addresses exactly this problem. It is obvious that the restriction to define an action only once can not be lifted without any additional effort. A mechanism needs to be introduced to determine which of the given actions can be applied to obtain a consistent successor situation. One way of doing so is to force the system to always prefer the most specific action description to the other descriptions of the same action. Informally an action description is more specific than another description of the same action, if the conditions of the latter are a proper subset of the conditions of the former. This method guarantees that whenever an applicable and most specific action is executed in a consistent situation the resulting situation is also consistent.

For more detailed information on specificity we want to refer to [HT93, HT95a], where a formal solution to the problem of specificity in the resource-oriented deductive approach ELP to action and change is presented. Furthermore this work contains the soundness and completeness results of SLDENF-resolution, since in order to solve the problem of specificity it turned out to be crucial to augment SLDE-resolution by negation as failure [Cla78].

2.3.4 Object–Oriented Aspects and ELP

An interesting extension to logic programming is the integration of principles used in object oriented programming. It offers the advantage of both paradigms, i.e. the declarativeness of logic programming, and — of object oriented programming — the structured representation of data in classes of objects, the inheritance principle among classes together with the dynamic modification of data. In [GHS+92] it is suggested to extend ELP by treating situations and actions as objects and methods respectively, where the properties of an object are represented as facts which are connected via the AC1–function symbol $\circ$. Communication between objects is realized by means of shared variables. In this manner the predicate $object(o, t, o')$ — in analogy to the ternary predicate $plan(s, p, s')$ — models objects with the intended meaning that the object o is transformed into object o' after receiving message t.

One of the most important principles in object oriented programming is the inheritance of methods from a class of objects to its subclasses. In [GHS+92] it has been shown, that within ELP methods defined for a certain class are automatically inherited by its subclasses, thus ELP incorporates the principle of inheritance. However, sometimes a method defined for a superclass needs to be modified or even changed completely in order to be applicable for a particular subclass. This is done by overriding the method inherited from the superclass, which makes it possible to break the inheritance principle in specific exceptional cases.

In [Thi94b, HT95a] there is a formalization introduced which shows that the concept of specificity (see Section 2.3.3) is suitable to model classes, objects and methods as well as inheritance and overriding. In particular, one can avoid inconsistent objects by using the basic ideas underlying specificity for generating the completion of a method, i.e. this involves finding more specific variants of a method for those methods which, when applied to an object, are likely to result in an inconsistent object.

In addition to the extension of ELP towards representing objects and methods in [GHS+92] it is pointed out that ELP can be further employed for modeling database transactions using a predicate $db(d, t, d')$, which can be interpreted declaratively as the execution of the transaction sequence t transforms the database d into the database d'.

2.3.5 The Relation of ELP to the Action Description Language $\mathcal{A}$ and the Ego-World-Semantics

At the moment there are two well-known semantics for the automation of reasoning about causality. One of them is the action description language $\mathcal{A}$ introduced by M. Gelfond and V. Lifschitz [GL93]. The other one was presented by E. Sandewall [San94] and is called *Ego-World-Semantics* (or EWS for short).

$\mathcal{A}$ is a formalism for describing dynamic systems in a simple, elegant and natural way. Compared with other such formalisms, it offers two interesting opportunities. On the basis of observations one can draw conclusions on past situations, such that situations do not necessarily need to be specified completely. In addition reasoning about hypothetical developments of the world becomes possible. The simplicity of $\mathcal{A}$ is based on the fact that it only allows constants from propositional logic for describing situations. A problem description in $\mathcal{A}$ consists of a general description of the effects of all actions defined for a specific scenario together with a description of concrete observations made during the performance of particular sequences of actions. The aim is to generate possible models, which explain these observations taking into account the general descriptions of the effects of actions. The execution of actions adds or removes elements from a particular set of facts representing some situation in the world. All facts not affected by the action continue to hold in the resulting situation due to the assumption of persistence, which says that everything in the world not affected by the execution of an action stays the same, i.e. persists.

The states of a system described in $\mathcal{A}$ are stable in that a system in a specific state only enters a new state if the execution of an action transforms the current state into some successor state. The sequence of states of a system described in $\mathcal{A}$ is determined by the initial state and the sequence of actions starting from it. Since $\mathcal{A}$ only allows the description of deterministic actions, the resulting sequence of states can be determined exactly.

In $\mathcal{A}$ one can find the set of initial states of a system by means of propositions about possible successor states resulting from the given sequences of actions. An evaluation of these propositions restricts the set of possible initial states to the set of those states which can be transformed into successor states for which these propositions hold. The semantics of $\mathcal{A}$ allows to draw conclusions on such propositions through the effects of the sequences of actions which can be performed starting from the initial state of a specific system.

Because of its simplicity and strength $\mathcal{A}$ became very popular and widely used. It was translated into other formalisms like extended logic programming [GL93], abductive logic programming [DdS93, Dun93], and equational logic programming [Thi94c] as well. It thereby turned out that ELP is an appropriate concept for such a translation of $\mathcal{A}$, it is sound and complete with respect to the semantics of $\mathcal{A}$. This result shows that the use of ELP is not only suitable for solving planning problems but postdiction problems (see Section 1.1) too, which establishes the expressive power of ELP.

The second semantical method to the formalization of situations, actions, and causality — the EWS — was developed by E. Sandewall. It provides features for describing the behavior of dynamic systems in which an agent, called *ego* acts and thereby changes the state of the whole system. The basic idea is to understand the development of the system-state as an interplay between the acting agent and the world, reacting on the actions performed by the agent. In this context one aims for finding models explaining the observations made about the current state of the system; by a model we mean a complete description of the development of the system.

In EWS states of a dynamic system are characterized by sets of features, and time is represented by natural numbers. The development of a dynamic system can be described by stating its state for each moment in time giving from its initial state. A complete description of this type is called history. As usual, the performance of actions causes the change of states of such a system. Actions in EWS are formally described by a table of their effects, containing for each possible state of the system the features affected as well as how they were affected.

The EWS is a method for proving the suitability of deductive formalisms for reasoning about causality for specific problem classes. It assumes that the formalization of the respective scenario is based on a complete description of the effects of actions. This is known as the *accurate knowledge* ($\mathcal{K}$) assumption. Furthermore, it is assumed that the part of the system description not affected by the performance of an action continues to hold afterwards, which is known as the persistence assumption or *inertia principle* (I). Last, it allows actions to have *alternative results* (A) depending on the state of the system when performing the action. Problem classes satisfying these assumptions are called $\mathcal{K}$-IA classes.

In [Thi94b] it has been shown that there are equivalences between the action description language $\mathcal{A}$ and the adequate problem classes in the EWS. Consequently, a method which was proved to be sound and complete with respect to either $\mathcal{A}$ or the mentioned problem classes of EWS, is also suitable for the corresponding part of the respective other concept. This result applies especially to the ELP–approach.

2.3.6 Concurrent and Non-Deterministic Actions

Concurrent actions play an important part in many complex dynamic systems. When formalizing a system that allows the concurrent execution of actions there arises the problem of that it is impractical to define the effects of the concurrent execution of each possible combination of actions explicitly. One would rather like to infer these effects from the partial descriptions of the involved individual actions. Unfortunately descriptions like this may sometimes lead to contradictory effects. The question is then how we should interpret these contradictions?

The language $\mathcal{A}_C$ introduced by C. Baral and M. Gelfond [BG93] is an extension of the action description language $\mathcal{A}$, which allows to describe concurrent actions. Within [BT94] there was a further extension of $\mathcal{A}_C$ introduced, viz. the dialect $\mathcal{A}_C^+$. This dialect makes it possible to infer non-contradicted information from contradictory descriptions. Furthermore, $\mathcal{A}_C^+$ allows to describe non-deterministic actions with randomized effects and uncertain knowledge. The outstanding achievement of [BT94] is the presentation of the first sound and complete translation from both $\mathcal{A}_C$ and $\mathcal{A}_C^+$ (by a simple modification) into a logic program with an underlying equational theory, which allows to automate reasoning about situations, actions, and change according to the concepts determined by $\mathcal{A}_C$ and $\mathcal{A}_C^+$. It turned out that an extension of the ELP–approach is suited as a sound and even complete method for encoding $\mathcal{A}_C$ and $\mathcal{A}_C^+$, which proves once again that ELP is an appropriate concept for formalizing reasoning about situations, actions, and causality.

The central emphasis within this work is to further extend the ELP–approach to be able to handle hierarchical planning problems. Our basic new idea is the reification of actions, i.e. actions as well as facts are treated as resources which can be produced and consumed. As mentioned in the introduction there are scenarios in which this point of view is essential for modeling them according to their real world domain. By means of such a new view on actions it becomes possible to synthesize a complex action from other less complex resources on a lower abstraction level. This is exactly where we see the starting point for modeling abstraction and hierarchical planning in ELP. But before starting to develop our new concept we want to introduce the main features of the ELP–approach in more detail.

2.4 The Equational Logic Programming Approach

Equational logic programming is an approach to logic programming which describes situations, states etc. as terms. The respective subterms of such a description can then be treated as resources. This work is based on a deductive, resource-oriented view on planning, using a slightly modified ELP–approach compared to the original ELP which we have introduced in Section 2.3.2.

In the original ELP–approach introduced in Section 2.3.2, the planning process was modeled with the help of a ternary predicate $plan(i,p,g)$ with intended meaning that the execution of plan p transforms the initial situation i into the goal situation g. Situations are modeled by a binary function $\circ$, which is associative (A), commutative (C), and admits a unit element $\emptyset$ (1). For example, the initial and goal situations in the tea example are represented by $d \circ q$ and $t \circ Z$, respectively, where Z is a variable expressing the fact that we do not care of other resources in the goal situation aside from the tea we strive for. In this work we will use the notation from [HT95a]. It differs from [HS90] in that the predicate *plan* is replaced by a predicate *causes* (see (2.6)), and the actions are explicitly stated using a predicate *action* (see (2.7)). While in [HT95a] the plan or action is represented by an additional argument, we will omit this argument in this work for brevity. Section 3.3 contains a detailed discussion on how a plan can be extracted from a proof.

$$\begin{array}{lll} causes(I,I). & & \\ causes(C \circ V, G) & \leftarrow & action(C,E) \wedge causes(E \circ V, G). \end{array} \tag{2.6}$$

In general the k actions belonging to the respective scenario are given by k unit clauses of the form

$$\begin{array}{c} action(c_1, e_1). \\ \vdots \\ action(c_k, e_k). \end{array} \tag{2.7}$$

where c_i and e_i ($1 \leq i \leq k$) are AC1–terms representing the conditions and effects of the actions respectively.

As $\circ$ is an AC1–function symbol, the respective equational axioms have to be built into the deductive machinery. Therefore, SLDE–resolution is used as the rule of inference for the equational logic program, i.e. SLD–resolution augmented by a special purpose unification procedure [Höl89].

In the example, the actions *get–change* and *get–tea* can now be represented by the following two unit clauses.

$$\begin{array}{l} action(d, q \circ q \circ q \circ q). \\ action(q \circ q \circ q, t). \end{array} \tag{2.8}$$

We can solve the planning problem by posing the query:

$$\leftarrow causes(d \circ q, t \circ Z).$$

The above query admits the SLDE–refutation (see below) wrt. $\{(2.6),(2.8)\}$ and the equational theory AC1 yielding the computed answer substitution $\{Z \mapsto q \circ q\}$. The plan can be extracted from this proof by means of recording the actions performed and consists of the actions *get–change* followed by *get–tea*.

$$\begin{array}{l} \leftarrow causes(d \circ q, t \circ Z). \\ \leftarrow action(d, E') \wedge causes(E' \circ q, t \circ Z). \\ \leftarrow causes(q \circ q \circ q \circ q \circ q, t \circ Z). \\ \leftarrow action(q \circ q \circ q, E'') \wedge causes(E'' \circ q \circ q, t \circ Z). \\ \leftarrow causes(t \circ q \circ q, t \circ Z). \\ \square \end{array}$$

The ELP–approach is purely declarative, admits the standard semantics of a logic program with an underlying equational theory, and is sound and complete [HS90]. Furthermore, the standard semantics corresponds to multiset rewriting steps [HT95a]; e.g., the initial situation of the exemplary tea scenario is modeled by the multiset $\{d, q\}$, and is rewritten to $\{q, q, q, q, q\}$ by the action get–change and further to $\{t, q, q\}$ by *get–tea*. In general, if $\mathcal{I}$ denotes a situation and $\langle \mathcal{C}.\mathcal{E} \rangle$ an action with conditions $\mathcal{C}$ and effects $\mathcal{E}$, then the action is applicable to $\mathcal{I}$ iff $\mathcal{C} \dot{\subseteq} \mathcal{I}$ and, if applied, yields the subsequent situation $(\mathcal{I} \dot{-} \mathcal{C}) \dot{\cup} \mathcal{E}$.

In the sequel we further assume that actions are ground. This will straighten out the representation. We will lift this restriction in the final Chapter 5.

2.5 The Chemical Abstract Machine

The chemical metaphor used for hierarchical planning in Section 3.2 is adapted from the Chemical Abstract Machine (CHAM) which was first introduced by G. Berry and G. Boudol to model asynchronous concurrent computations [BB90].

The states of the CHAM are chemical solutions where floating molecules can interact freely according to interaction rules; a magical mechanism stirs the solution, allowing for possible contacts between molecules. The basic methodology underlying the CHAM is liberated from control management, i.e. the concurrent components are freely moving in the system and interact when they come in contact. The transformation process of the solution is obviously parallel since any number of reactions can be performed in parallel provided that they involve disjoint sets of molecules. Solutions can be stratified by encapsulating subsolutions within a membrane that forces reactions to occur locally.

Formally a CHAM is specified by molecules $m, m', \ldots$, solutions, i.e. multisets of molecules, $S, S', \ldots$ and a set of interaction rules. The molecules are terms of an algebra and the solutions are finite multisets of molecules, written $\{\!\!\{ m_1, m_2, \ldots, m_k \}\!\!\}$. Since chemical concurrency is naturally associative and commutative the representation of solutions by multisets of molecules is a reasonable choice. In each CHAM any solution S can itself be considered as a single molecule and can thus appear as a subsolution. The corresponding operator $\{\!\!\{\}\!\!\}$ is called the membrane operator.

There are in general two kinds of molecules, some of which do not exhibit interaction capabilities, while others — the so called *ions* — are ready to interact. A solution can be heated to break complex molecules into smaller molecules up to ions. In order to rebuild complex molecules from components a solution can be cooled.

The interaction rules of a CHAM are multiset rewritings of the form $m_1, m_2, \ldots, m_k \rightarrow m'_1, m'_2, \ldots, m'_l$, where m_i and m_j are molecules; they are presented as *rule schemata*, and the actual rules are instances of these schemata. The rules determine an interaction relation $S \rightarrow S'$ between solutions. All interactions obey the following four laws:

- The Reaction Law. An instance of the right-hand-side of a rule can replace the corresponding instance of its left-hand-side. If

$$m_1, m_2, \ldots, m_k \rightarrow m'_1, m'_2, \ldots, m'_l$$

is a rule and $M_1, M_2, \ldots, M_k, M'_1, M'_2, \ldots, M'_l$, are instances of the m_i's and the m_j's, then

$$\{\!\mid M_1, M_2, \ldots, M_k \mid\!\} \rightarrow \{\!\mid M'_1, M'_2, \ldots, M'_l \mid\!\}. \tag{L1}$$

- The Chemical Law. Reactions can be performed freely within any solution:

$$\frac{S \rightarrow S'}{S \dot{\cup} S'' \rightarrow S' \dot{\cup} S''} \tag{L2}$$

- The Membrane Law. A subsolution can evolve freely in any context[6]:

$$\frac{S \rightarrow S'}{\{\!\mid C[S] \mid\!\} \rightarrow \{\!\mid C[S'] \mid\!\}} \tag{L3}$$

In addition some CHAMs use the airlock-construct. An airlock is a molecule of the form $m \triangleleft S$ where m is a molecule and S is a solution. Airlocks are built and suppressed by the following law:

- The Airlock Law.

$$\{\!\mid m \mid\!\} \dot{\cup} S \leftrightarrow \{\!\mid m \triangleleft S \mid\!\} \tag{L4}$$

Each individual CHAM is given by specifying a particular set of transformation rules that determine interactions between molecules such that the transformations of solutions obey the above mentioned laws.

One of the illustrative examples of a CHAM given in [BB90] is a fragment CCS$^-$ of Milner's process calculus CCS [Mil89]. Since we want to follow this specific CHAM as closely as possible, we are next going to introduce the syntax of this machine together with its set of transformation rules. Let $\mathcal{N} = \{a, b, \ldots\}$ be a set of names and $\mathcal{L} = \{a^+, a^-\} \mid a \in \mathcal{N}\}$ be a set of labels. The symbols $\alpha, \beta, \ldots$ are used to range over labels. There are four basic operators, which are '0' (inaction), '$.$' (prefixing), '$\mid$' (parallel) and '$\backslash$' (restriction).

Molecules are of the following form:

$$\begin{aligned} m \quad &::= \quad p \mid \alpha.m \mid m\backslash a \mid S \mid m \triangleleft S \\ \text{where } p \quad &::= \quad 0 \mid \alpha.p \mid p|p \mid p\backslash a. \end{aligned}$$

[6] We use the context notation $C[]$ — as in the λ-calculus [Bar81] — to denote a molecule with a hole $[]$ in which to place another molecule.

Ions are molecules of the form $\alpha.p$, where α — the valence — denotes the ion's communication capability. From the above syntax-definition it can be seen that a solution can be considered as single molecule and vice versa.

The set of transformation rules contains the following:

$$
\begin{array}{rcll}
p_1|p_2 & \rightleftharpoons & p_1, p_2 & \text{(parallel)} \\
a^+.p_1, a^-.p_2 & \rightarrow & p_1, p_2 & \text{(reaction)} \\
0 & \rightharpoonup & & \text{(inaction cleanup)} \\
(\alpha.p)\backslash a & \rightleftharpoons & \alpha.(p\backslash a) \text{ if } \alpha \notin \{a^+, a^-\} & \text{(restriction ion)} \\
p\backslash a & \rightleftharpoons & \{\!|p|\!\}\backslash a & \text{(restriction membrane)} \\
\{\!|m, m_1, m_2, \ldots, m_n|\!\} & \rightleftharpoons & \{\!|m \lhd \{\!|m_1, m_2, \ldots, m_n|\!\}|\!\} & \text{(airlock)} \\
(\alpha.p) \lhd S & \rightleftharpoons & \alpha.(p \lhd S) & \text{(heavy ion)}
\end{array}
$$

The parallel rule says, that by heating ($\rightharpoonup$) a molecule of the form $p_1|p_2$ it breaks up into its components p_1 and p_2. Conversely, by cooling down ($\leftharpoondown$) a pair of molecules p_1, p_2 the compound molecule $p_1|p_2$ can be rebuilt. Only *ions*, which are molecules of the form $\alpha.p$, where α represents the valence of the ion, are allowed to perform reactions according to the reaction rule. A reaction is an irreversible transformation in which two complementary ions, floating in a solution, react with each other, whereby they release their bodies and the valences vanish. The inaction cleanup rule simply states that 0 evaporates when heated. The restriction ion and restriction membrane rules are special heating and cooling rules for restricted molecules. In the airlock rule the airlock mechanism is specified and the heavy ion rule denotes how to build an ion out of a molecule in an airlock.

In order to see the Chemical Abstract Machine at work, let us consider a simple execution, e.g. of the molecule $a^-.0|b^+.a^-.0|b^-.0$ [7]:

$$
\begin{array}{ccl}
 & \{\!|\ \underline{a^-.0|b^+.a^+.0|b^-.0}\ |\!\} & \\
\stackrel{*}{\rightharpoonup} & \{\!|\ a^-.0,\ \underline{b^+.a^+.0},\ \underline{b^-.0}\ |\!\} & \text{(parallel)} \\
\rightarrow & \{\!|\ \underline{a^-.0},\ \underline{a^+.0},\ 0\ |\!\} & \text{(reaction)} \\
\rightarrow & \{\!|\ \underline{0}, \underline{0}, \underline{0}\ |\!\} & \text{(reaction)} \\
\stackrel{*}{\rightharpoonup} & \{\!|\ |\!\} & \text{(inaction cleanup)}
\end{array}
$$

The execution of a solution starts from its initial state. By heating the molecules in the solution the potential communications can be released. They will then be used to perform reactions. A hot solution obtained by heating a solution as much as possible contains only ions, whereas any solution produced by transitions from the initial solution can be frozen by cooling rules.

[7] By $*$ we denote the repeated application of an interaction rule.

In general, chemical executions are non-deterministic. E.g. consider the execution of a solution

$$\{\, b^{+}.0, b^{-}.a^{+}.0, b^{-}.c^{+}.0 \,\}.$$

The $b^{+}.0$ ion can react with any of the two other ions. A reaction with the first yields $\{\, 0, a^{+}.0, b^{-}.c^{+}.0 \,\}$, and with the second $\{\, 0, b^{-}.a^{+}.0, c^{+}.0 \,\}$.

Chapter 3

A Resource-Oriented Deductive Approach for Hierarchical Planning

3.1 Resources

In this section we formally introduce and illustrate the new approach. We start out with the previously mentioned tea example to give the reader an idea of how the new concept is actually working. Thereafter, we extend it by incorporating generators. Finally, we show that the new approach developed so far forms a conservative extension the old one.

3.1.1 Actions as Resources

The ELP–approach models actions as if they were always available. This is due to the fact that the actions are represented by predicates. As emphasized in the introduction, there are real world situations in which actions are not properties, which hold forever, rather they are resources themselves. Hence, to increase the expressive power of the ELP–approach, and, as we will see in Section 3.2, to enable us to model hierarchical planning in a purely declarative way, we extend ELP and represent actions as resources.

We start by showing that we can still solve the tea problem even if the actions are modeled as resources. Formally, an action is denoted by a tuple $\langle \mathcal{C}.\mathcal{E} \rangle$ where $\mathcal{C}$ and $\mathcal{E}$ are multisets of terms of the form $\emptyset$ or $\{\!\{ t_1, \ldots, t_n \}\!\}$ [1], $n \geq 1$, representing *conditions* and *effects* of an action or sequence of actions, respectively. Facts about a situation can also be represented as actions, viz. as tuples of the form $\langle \emptyset.\mathcal{E} \rangle$; hereafter we want to call these expressions *primitive agents*, and all other expressions of the above type $\langle \mathcal{C}.\mathcal{E} \rangle$, *agents*. A *situation* is now simply a multiset of agents.

[1] The terms t_i, $1 \leq i \leq n$, are "simple" in that they do not contain representations of multisets nor are they representations of multisets themselves.

On the highest level of abstraction, the tea example can be formalized in the following way:[2]

$$\begin{array}{c} \text{Initial situation:} \\ \{\ \langle \emptyset.\{d\}\rangle,\ \langle \emptyset.\{q\}\rangle,\ \langle \{d\}.\{q,q,q,q\}\rangle,\ \langle \{q,q,q\}.\{t\}\rangle\ \} \\ \Downarrow \\ \textbf{?} \\ \Downarrow \\ \text{Goal situation:} \\ \{\ \langle \emptyset.\{t\} \mathbin{\dot\cup} Z\rangle\ \} \mathbin{\dot\cup} S \end{array} \tag{3.1}$$

Here, the initially available dollar and quarter are represented by the primitive agents $\langle \emptyset.\{d\}\rangle$ and $\langle \emptyset.\{q\}\rangle$, respectively, while the change and vending machines are represented by $\langle \{d\}.\{q,q,q,q\}\rangle$ and $\langle \{q,q,q\}.\{t\}\rangle$, respectively. The variables Z and S occurring in the goal situation represent the other effects different from tea possibly resulting from the transformation and the remaining other agents possibly still floating in the solution, respectively.

In order to solve the planning problem we have to state how the agents interact. Two agents $\langle \mathcal{C}_1.\mathcal{E}_1\rangle$ and $\langle \mathcal{C}_2.\mathcal{E}_2\rangle$ with $\mathcal{D} \mathrel{\dot\subseteq} \mathcal{C}_2 \mathbin{\dot\cap} \mathcal{E}_1$, for some $\mathcal{D} \mathrel{\dot\neq} \{\}$, contained in a solution can react with each other producing a new agent $\langle \mathcal{C}.\mathcal{E}\rangle$ with $\mathcal{C} = \mathcal{C}_1 \mathbin{\dot\cup} (\mathcal{C}_2 \mathbin{\dot-} \mathcal{D})$ and $\mathcal{E} = \mathcal{E}_2 \mathbin{\dot\cup} (\mathcal{E}_1 \mathbin{\dot-} \mathcal{D})$. The two agents $\langle \mathcal{C}_1.\mathcal{E}_1\rangle$ and $\langle \mathcal{C}_2.\mathcal{E}_2\rangle$ are consumed during the reaction. E.g., $\langle \emptyset.\{q\}\rangle$ and $\langle \{q,q,q\}.\{t\}\rangle$ can react yielding $\langle \{q,q\}.\{t\}\rangle$.

We can now find suitable reactions to replace the "?" in transformation (3.1), where we have underlined the interacting agents.

$$\begin{array}{c} \text{Initial situation:} \\ \{\ \underline{\langle \emptyset.\{d\}\rangle},\ \langle \emptyset.\{q\},\ \underline{\langle \{d\}.\{q,q,q,q\}\rangle},\ \langle \{q,q,q\}.\{t\}\rangle\ \} \\ \Downarrow \\ \{\ \langle \emptyset.\{q\}\rangle,\ \underline{\langle \emptyset.\{q,q,q,q\}\rangle},\ \underline{\langle \{q,q,q\}.\{t\}\rangle}\ \} \\ \Downarrow \\ \text{Final situation matching the goal situation in (3.1):} \\ \{\ \langle \emptyset.\{q\}\rangle,\ \langle \emptyset.\{t,q\}\rangle\ \} \end{array} \tag{3.2}$$

The result of matching[3] the goal situation stated in (3.1) with the final situation obtained in (3.2) yields the answer substitution $\{Z \mapsto \{q\},\ S \mapsto \{\langle \emptyset.\{q\}\rangle\}\}$. Obviously the plan which transforms the initial situation into the goal situation can be extracted from transformation (3.2). It consists of the consecutive actions *get–change* and *get–tea*, since we first consumed the action $\langle \{d\}.\{q,q,q,q\}\rangle$ followed by the action $\langle \{q,q,q\}.\{t\}\rangle$.

This is not the only solution to the planning problem stated in (3.1). Alternatively, Bert could insert his initial quarter into the vending machine, change the dollar note, and then insert two of the four quarters he obtained, thus activating the action *get–tea* which provides the tea he wanted. This can be

[2] The symbol "$\Downarrow$" represents the transition from one solution to another whereby the latter is achieved by an interaction or a series of interactions within the former.

[3] Modulo multiset operations.

represented by the following transformation:

$$\begin{array}{c}
\text{Initial situation:} \\
\{\ \langle\emptyset.\{d\}\rangle,\ \underline{\langle\emptyset.\{q\}\rangle},\ \langle\{d\}.\{q,q,q,q\}\rangle,\ \underline{\langle\{q,q,q\}.\{t\}\rangle}\ \} \\
\Downarrow \\
\{\ \underline{\langle\emptyset.\{d\}\rangle},\ \underline{\langle\{d\}.\{q,q,q,q\}\rangle},\ \langle\{q,q\}.\{t\}\rangle\ \} \\
\Downarrow \\
\{\ \underline{\langle\emptyset.\{q,q,q,q\}\rangle},\ \underline{\langle\{q,q\}.\{t\}\rangle}\ \} \\
\Downarrow \\
\text{Final situation matching the goal situation in (3.1):} \\
\{\ \langle\emptyset.\{t,q,q\}\rangle\ \}
\end{array} \tag{3.3}$$

What we have achieved above is a perfect solution which may well occur in the real world. One should observe that this solution cannot be modeled within the original ELP–approach as there actions are only applicable if all of their conditions are satisfied, whereas in the new approach actions are applicable as soon as their conditions are partially satisfied.

Unfortunately, the new approach does not yet work in general. Suppose Bert wants to buy another tea for his girlfriend Sue. This would be a problem if we represented the initial situation as we did before. Not only does Bert not have enough money, but also we have consumed the agents representing the actions *get–change* and *get–tea*, i.e. the vending machine might be out of tea bags and the change machine out of quarters, which means that somebody has to come and restock them.

To solve the problem, we may assume that we know the current amount of tea bags and quarters available in each machine, e.g. they contain three tea bags and eight quarters respectively. We therefore start the transformation with an initial situation containing three agents $\langle\{q,q,q\}.\{t\}\rangle$ and two agents $\langle\{d\}.\{q,q,q,q\}\rangle$.

$$\begin{array}{c}
\text{Initial situation:} \\
\{\ \underline{\langle\emptyset.\{d\}\rangle},\ \langle\emptyset.\{q\}\rangle,\ \underline{\langle\{d\}.\{q,q,q,q\}\rangle},\ \langle\{d\}.\{q,q,q,q\}\rangle,\ \langle\{q,q,q\}.\{t\}\rangle,\ \langle\{q,q,q\}.\{t\}\rangle,\ \langle\{q,q,q\}.\{t\}\rangle\ \} \\
\Downarrow \\
\{\ \langle\emptyset.\{q\}\rangle,\ \underline{\langle\emptyset.\{q,q,q,q\}\rangle},\ \langle\{d\}.\{q,q,q,q\}\rangle,\ \underline{\langle\{q,q,q\}.\{t\}\rangle},\ \langle\{q,q,q\}.\{t\}\rangle,\ \langle\{q,q,q\}.\{t\}\rangle\ \} \\
\Downarrow \\
\text{Final situation matching the goal situation in (3.1):} \\
\{\ \langle\emptyset.\{q\}\rangle,\ \langle\{d\}.\{q,q,q,q\}\rangle,\ \langle\emptyset.\{t,q\}\rangle,\ \langle\{q,q,q\}.\{t\}\rangle,\ \langle\{q,q,q\}.\{t\}\rangle\ \}
\end{array}$$

If Bert now had another quarter he could buy the tea for Sue. He could even change another dollar to buy it.

3.1.2 Generators

In general, however, if we want the new approach to be a conservative extension of the ELP–approach, we need to be able to model an unlimited supply of actions. In the example, this corresponds to an abstraction: We may decide not to care of the remaining quarters and tea–bags in the change and vending machine, respectively, and to assume that there is an unlimited supply of these resources. Recall that a situation is modeled by a multiset of agents. Hence, an unlimited supply of a certain agent can be modeled by allowing such an agent to be duplicated. This is a form of controlled weakening and was first introduced by J.Y. Girard within linear logic [Gir87]. There, so–called exponentials are employed to copy resources.

We too will use an exponential, written "!", which, placed in front of an agent $\langle \mathcal{C}.\mathcal{E} \rangle$, turns the agent into a new type of agent $!\langle \mathcal{C}.\mathcal{E} \rangle$, called a *generator*. Such a generator can be used to generate a new agent–molecule, i.e. $!\langle \mathcal{C}.\mathcal{E} \rangle$ may be replaced by $\langle \mathcal{C}.\mathcal{E} \rangle, !\langle \mathcal{C}.\mathcal{E} \rangle$ within a multiset, which can then interact with other molecules. One should note that a generator must not be used directly in a reaction.

In the tea example we can now specify the initial situation by using generators instead of agents to represent the *get–change* and *get–tea* actions. The following transformation illustrates the use of generators.

$$
\begin{array}{c}
\text{Initial situation:} \\
\{\ \langle \emptyset.\{d\}\rangle,\ \langle \emptyset.\{q\}\rangle,\ \underline{!\langle \{d\}.\{q,q,q,q\}\rangle},\ !\langle \{q,q,q\}.\{t\}\rangle\ \} \\
\Downarrow \\
\{\ \underline{\langle \emptyset.\{d\}\rangle},\ \langle \emptyset.\{q\}\rangle,\ \underline{\langle \{d\}.\{q,q,q,q\}\rangle},\ !\langle \{d\}.\{q,q,q,q\}\rangle,\ !\langle \{q,q,q\}.\{t\}\rangle\ \} \\
\Downarrow \\
\{\ \langle \emptyset.\{q\}\rangle,\ \langle \emptyset.\{q,q,q,q\}\rangle,\ !\langle \{d\}.\{q,q,q,q\}\rangle,\ \underline{!\langle \{q,q,q\}.\{t\}\rangle}\ \} \\
\Downarrow \\
\{\ \langle \emptyset.\{q\}\rangle,\ \underline{\langle \emptyset.\{q,q,q,q\}\rangle},\ !\langle \{d\}.\{q,q,q,q\}\rangle,\ \underline{\langle \{q,q,q\}.\{t\}\rangle},\ !\langle \{q,q,q\}.\{t\}\rangle\ \} \\
\Downarrow \\
\text{Final situation matching the goal situation in (3.1):} \\
\{\ \langle \emptyset.\{q\}\rangle,\ !\langle \{d\}.\{q,q,q,q\}\rangle,\ \langle \emptyset.\{t,q\}\rangle,\ !\langle \{q,q,q\}.\{t\}\rangle\ \}
\end{array}
$$

After the transformation, we obtain a situation which still contains the two generators representing the prevailing source of *get–change* and *get–tea* actions. This way it is possible to handle resources which are always available.

3.1.3 The New Approach

After having presented the intended semantics of the new approach to planning, where actions are modeled as resources, we will now specify an equational logic program which gives us an equivalent operational semantics.

First of all, there is no need to change the equational part. Multisets of agents are represented with the help of the AC1–function symbol $\circ$, and the respective equational axioms are built into the unification computation. The facts about a situation as well as the conditions and effects of actions are represented as "simple" terms, i.e. terms which do not contain denotations for multisets. In what follows, the predicate $S =_{AC1} T$ is used to explicitly denote that S and T are to be unified modulo the AC1–theory.

The multiset rewritings used in the intended semantics are modeled with the help of a binary predicate $transform(i, g)$, which holds if the initial situation i is transformed into goal situation g .

$$
\begin{array}{lcl}
transform(I, I). & & \\
transform(I, G) & \leftarrow & interact(I, S) \wedge transform(S, G).
\end{array}
\tag{3.4}
$$

The interactions itself are now not only applications of actions (if some of their conditions are satisfied), but also the duplication of actions specified by generators.

$$\begin{array}{l} interact(\langle C_1.E_1 \circ D\rangle \circ \langle C_2 \circ D.E_2\rangle \circ Z,\ \langle C_1 \circ C_2.E_1 \circ E_2\rangle \circ Z\rangle) \leftarrow D \neq_{AC1} \emptyset. \\ interact(!\langle C.E\rangle \circ S, \langle C.E\rangle \circ !\langle C.E\rangle \circ S). \end{array} \tag{3.5}$$

One should observe that we now need to apply SLDENF–resolution [She92, HT95a, Thi95b] as we have to check whether $D \neq_{AC1} \emptyset$ in the first clause of (3.5), i.e. we employ the *negation-as-failure* principle [Cla78] to handle negative subgoals. Furthermore, recall that actions are assumed to be ground. Hence, generators are ground and the duplication of an action within a generator does not pose any problems concerning variable renamings. This problem will be discussed in Section 5.1 when we lift the groundedness condition on actions.

It is easy to see that the new program consisting of the clauses in (3.4) and (3.5) can solve all planning problems discussed so far in this section. For example, consider (3.3). The corresponding SLDENF–refutation,

$$\begin{array}{l} \leftarrow transform(\langle\emptyset.d\rangle \circ \langle\emptyset.q\rangle \circ \langle d.q \circ q \circ q \circ q\rangle \circ \langle q \circ q \circ q.t\rangle, \langle\emptyset.t \circ Z\rangle \circ S) \\ \leftarrow interact(\langle\emptyset.d\rangle \circ \underline{\langle\emptyset.q\rangle} \circ \langle d.q \circ q \circ q \circ q\rangle \circ \underline{\langle q \circ q \circ q.t\rangle}, T') \wedge transform(T', \langle\emptyset.t \circ Z\rangle \circ S) \\ \leftarrow transform(\langle\emptyset.d\rangle \circ \langle d.q \circ q \circ q \circ q\rangle \circ \langle q \circ q.t\rangle, \langle\emptyset.t \circ Z\rangle \circ S) \\ \leftarrow interact(\underline{\langle\emptyset.d\rangle} \circ \underline{\langle d.q \circ q \circ q \circ q\rangle} \circ \langle q \circ q.t\rangle, T'') \wedge transform(T'', \langle\emptyset.t \circ Z\rangle \circ S) \\ \leftarrow transform(\langle\emptyset.q \circ q \circ q \circ q\rangle \circ \langle q \circ q.t\rangle, \langle\emptyset.t \circ Z\rangle \circ S) \\ \leftarrow interact(\underline{\langle\emptyset.q \circ q \circ q \circ q\rangle} \circ \underline{\langle q \circ q.t\rangle}, T''') \wedge transform(T''', \langle\emptyset.t \circ Z\rangle \circ S) \\ \leftarrow transform(\langle\emptyset.t \circ q \circ q\rangle, \langle\emptyset.t \circ Z\rangle \circ S) \\ \square \end{array} \tag{3.6}$$

where we have underlined the two agents which interact as before, yields the answer substitution $\{S \mapsto \emptyset, Z \mapsto q \circ q\}$ as intended.

Moreover, the new program is a conservative extension of the original ELP–approach. To prove this claim formally, let the ELP–planning problem contain k actions of the form $action(c_i, e_i)$ with $1 \leq i \leq k$. In the new version, each of these actions will be encoded as a generator $!\langle c_i.e_i\rangle$. Furthermore, let $i_1, \ldots, i_m$ and $g_1, \ldots, g_n$ denote the resources available in the initial and goal situation, respectively.

Theorem 1 *If there exists an SLDE–refutation for*

$$\{\leftarrow causes(i_1 \circ \ldots \circ i_m, g_1 \circ \ldots \circ g_n \circ Y)\} \cup \{(2.6), (2.7)\}$$

then there exists an SLDENF–refutation for

$$\{\leftarrow transform(\langle\emptyset.i_1\rangle \circ \ldots \circ \langle\emptyset.i_m\rangle \circ !\langle c_1.e_1\rangle \circ \ldots \circ !\langle c_k.e_k\rangle, \langle\emptyset.g_1 \circ Z_1\rangle \circ \ldots \circ \langle\emptyset.g_n \circ Z_n\rangle \circ Z)\} \cup \{(3.4), (3.5)\}$$

Proof
The proof can be obtained by transfinite induction on the length l of the SLDE–refutation[4].

Suppose the theorem holds for all SLDE–refutations of length less than l. Let

$$\leftarrow \mathit{causes}(i_1 \circ \ldots \circ i_m, g_1 \circ \ldots \circ g_n \circ Y) \xrightarrow{l}_{\text{SLDE}} \square$$

be an SLDE–refutation of length l. Depending on the clause used in the first refutation step we distinguish the following two cases.

1. If the first clause of (2.6) was chosen, then $l = 1$ and the initial goal succeeds immediately. Hence, we find an AC1-unifier σ for $i_1 \circ \ldots \circ i_m$ and $g_1 \circ \ldots \circ g_n \circ Y$. Let $\theta = \{x \mapsto t \in \sigma \mid x \neq Y\}$ and assume that Y is bound to a term s in σ, i.e. $\sigma = \theta \cup \{Y \mapsto s\}$.

   Obviously, $\theta \cup \{Z \mapsto s \circ !\langle c_1.e_1\rangle \circ \ldots \circ !\langle c_k.e_k\rangle\} \cup \{Z_i \mapsto \emptyset \mid 1 \leq i \leq n\}$ is an AC1-unifier for $\langle \emptyset.i_1\rangle \circ \ldots \circ \langle \emptyset.i_m\rangle \circ !\langle c_1.e_1\rangle \circ \ldots \circ !\langle c_k.e_k\rangle$ and $\langle \emptyset.g_1 \circ Z_1\rangle \circ \ldots \circ \langle \emptyset.g_n \circ Z_n\rangle \circ Z$. Thus, there is an SLDENF–refutation of length l of the goal

   $$\leftarrow \mathit{transform}(\langle \emptyset.i_1\rangle \circ \ldots \circ \langle \emptyset.i_m\rangle \circ !\langle c_1.e_1\rangle \circ \ldots \circ !\langle c_k.e_k\rangle, \langle \emptyset.g_1 \circ Z_1\rangle \circ \ldots \circ \langle \emptyset.g_n \circ Z_n\rangle \circ Z) \quad (3.7)$$

   using the first clause of (3.4) as input clause.

2. If the second clause of (2.6) was chosen, we obtain an SLDE–refutation of the form

   $$\begin{array}{ll} \leftarrow \mathit{causes}(i_1 \circ \ldots \circ i_m, g_1 \circ \ldots \circ g_n \circ Y) & (3.8) \\ \leftarrow \mathit{action}(i_p \circ \ldots \circ i_q, E) \wedge \mathit{causes}(i_1 \circ \ldots \circ i_{p-1} \circ i_{q+1} \circ \ldots \circ i_m \circ E, g_1 \circ \ldots \circ g_n \circ Y) & (3.9) \\ \leftarrow \mathit{causes}(i_1 \circ \ldots \circ i_{p-1} \circ i_{q+1} \circ \ldots \circ i_m \circ e_j, g_1 \circ \ldots \circ g_n \circ Y) & (3.10) \\ \vdots & \\ \square & \end{array}$$

   where σ_1, σ_2 are the AC1-unifiers obtained in the first and second resolution step of the refutation. Let the resources $i_p \circ \ldots \circ i_q$, $1 \leq p \leq q \leq m$ be contained in the initial situation $s = i_1 \circ \ldots \circ i_m$. Furthermore, let $\mathit{causes}(C \circ V, G) \leftarrow \mathit{action}(C, E) \wedge \mathit{causes}(E \circ V, G)$ be the program clause used in the first refutation step. The unifier σ_1 is of the form

   $$\begin{array}{rllll} \sigma_1 & = \{ & C & \mapsto & i_p \circ \ldots \circ i_q, \\ & & V & \mapsto & i_1 \circ \ldots \circ i_{p-1} \circ i_{q+1} \circ \ldots \circ i_m, \\ & & G & \mapsto & g_1 \circ \ldots \circ g_n \circ Y \}, \end{array}$$

   and we obtain (3.9) from (3.8). Next we select the literal $\mathit{action}(i_p \circ \ldots \circ i_q, E)$ from (3.9). Let $\mathit{action}(c_j, e_j)$, $1 \leq j \leq k$ be the unit clause contained in (2.7) used to derive (3.10) from (3.9). Recall that the terms $i_p, \ldots, i_q$ and c_j denote resources, they are assumed to be ground. From the SLDE-refutation it follows that $i_p \circ \ldots \circ i_q =_{AC1} c_j$. Consequently, the unifier σ_2 is of the form $\{E \mapsto e_j\}$.

   Now, the term $i_1 \circ \ldots \circ i_{p-1} \circ i_{q+1} \circ \ldots \circ i_m \circ e_j$ represents the resources available in the new situation s' after the application of the action $\mathit{action}(c_j, e_j)$ to the initial situation s, which consumed the

[4] Without loss of generality we assume hereafter that the goals are processed from left to right.

resources $i_p \circ \ldots \circ i_q$ and produced the resources represented by the term e_j. Its corresponding term in the new approach contains a primitive agent for each resource, thus the corresponding situation $\hat{s'}$ is represented by the term $\langle\emptyset.i_1\rangle \circ \ldots \circ \langle\emptyset.i_{p-1}\rangle \circ \langle\emptyset.i_{q+1}\rangle \circ \ldots \circ \langle\emptyset.i_m\rangle \circ \langle\emptyset.e_j\rangle \circ !\langle c_1.e_1\rangle \circ \ldots \circ !\langle c_k.e_k\rangle$. Obviously, the term representing the goal situation was not yet affected by the refutation, thus it remains unchanged.

Now, the refutation of (3.10) is of length $l-2$ and we can apply the induction hypothesis which allows to conclude that there is an SLDENF–refutation of the form

$$\begin{array}{l} \leftarrow \mathit{transform}(\langle\emptyset.i_1\rangle \circ \ldots \circ \langle\emptyset.i_{p-1}\rangle \circ \langle\emptyset.i_{q+1}\rangle \circ \ldots \circ \langle\emptyset.i_m\rangle \circ \langle\emptyset.e_j\rangle \circ !\langle c_1.e_1\rangle \circ \ldots \circ !\langle c_k.e_k\rangle, \\ \qquad\qquad \langle\emptyset.g_1 \circ Z_1\rangle \circ \ldots \circ \langle\emptyset.g_n \circ Z_n\rangle \circ Z) \\ \vdots \\ \Box \end{array} \tag{3.11}$$

It is left to show how to obtain the remaining part of the SLDENF–refutation, i.e. how to derive the start–goal of (3.11) from the initial goal (3.7). From the SLDE–refutation we know that the difference between the two situations s and s' is the application of the action $\mathit{action}(c_j, e_j)$ to s. In order to apply an action in the new approach one first needs to generate this action. The generator for the action represented by the unit clause $\mathit{action}(c_j, e_j)$ is denoted by $!\langle c_j.e_j\rangle$ and since $1 \leq j \leq k$ it is contained in the term $\langle\emptyset.i_1\rangle \circ \ldots \circ \langle\emptyset.i_m\rangle \circ !\langle c_1.e_1\rangle \circ \ldots \circ !\langle c_k.e_k\rangle$ representing the initial situation $\hat{s}$ in the goal (3.7). Thus the next two steps in the SLDENF–derivation can be obtained using the following variants of the second clause of (3.4)

$$\mathit{transform}(I, G) \leftarrow \mathit{interact}(I, T) \wedge \mathit{transform}(T, G). \tag{3.12}$$

and the second clause of (3.13)

$$\mathit{interact}(!\langle C.E\rangle \circ S, \langle C.E\rangle \circ !\langle C.E\rangle \circ S). \tag{3.13}$$

Obviously we can find unifiers θ_1, θ_2 by first unifying the head of the clause in (3.12) with the initial goal (3.7), performing a derivation step yielding a consecutive goal containing a literal which is unifiable with the head of the clause in (3.13), respectively. They are of the form

$$\begin{array}{rlcl} \theta_1 = \{ & I & \mapsto & \langle\emptyset.i_1\rangle \circ \ldots \circ \langle\emptyset.i_m\rangle \circ !\langle c_1.e_1\rangle \circ \ldots \circ !\langle c_j.e_j\rangle \circ \ldots \circ !\langle c_k.e_k\rangle, \\ & G & \mapsto & \langle\emptyset.g_1 \circ Z_1\rangle \circ \ldots \circ \langle\emptyset.g_n \circ Z_n\rangle \circ Z \ \} \end{array}$$

and

$$\begin{array}{rlcl} \theta_2 = \{ & C & \mapsto & c_j, \\ & E & \mapsto & e_j, \\ & S & \mapsto & \langle\emptyset.i_1\rangle \circ \ldots \circ \langle\emptyset.i_m\rangle \circ !\langle c_1.e_1\rangle \circ \ldots \circ !\langle c_{j-1}.e_{j-1}\rangle \circ !\langle c_{j+1}.e_{j+1}\rangle \circ \ldots \circ !\langle c_k.e_k\rangle, \\ & T & \mapsto & \langle c_j.e_j\rangle \circ !\langle c_j.e_j\rangle \circ S \ \}. \end{array}$$

Hence, we obtain the following part of the SLDENF–derivation

$$\begin{array}{l} \leftarrow \mathit{transform}(\langle\emptyset.i_1\rangle \circ \ldots \circ \langle\emptyset.i_m\rangle \circ !\langle c_1.e_1\rangle \circ \ldots \circ !\langle c_j.e_j\rangle \circ \ldots \circ !\langle c_k.e_k\rangle, \\ \qquad\qquad \langle\emptyset.g_1 \circ Z_1\rangle \circ \ldots \circ \langle\emptyset.g_n \circ Z_n\rangle \circ Z) \\ \leftarrow \mathit{interact}(\langle\emptyset.i_1\rangle \circ \ldots \circ \langle\emptyset.i_m\rangle \circ !\langle c_1.e_1\rangle \circ \ldots \circ !\langle c_j.e_j\rangle \circ \ldots \circ !\langle c_k.e_k\rangle, T) \wedge \\ \quad \mathit{transform}(T, \langle\emptyset.g_1 \circ Z_1\rangle \circ \ldots \circ \langle\emptyset.g_n \circ Z_n\rangle \circ Z) \\ \leftarrow \mathit{transform}(\langle\emptyset.i_1\rangle \circ \ldots \circ \langle\emptyset.i_m\rangle \circ \langle c_j.e_j\rangle \circ !\langle c_1.e_1\rangle \circ \ldots \circ !\langle c_j.e_j\rangle \circ \ldots \circ !\langle c_k.e_k\rangle, \\ \qquad\qquad \langle\emptyset.g_1 \circ Z_1\rangle \circ \ldots \circ \langle\emptyset.g_n \circ Z_n\rangle \circ Z) \end{array} \tag{3.14}$$

In the SLDE-refutation the resources $i_p \circ \ldots \circ i_q$ were consumed in *one* derivation step. Recall that for each of these resources i_l , $p \leq l \leq q$ there is a primitive agent $\langle \emptyset . i_l \rangle$ contained in $\hat{s}$. As we mentioned earlier on, only two agents are allowed to participate in a reaction. Consequently, in order to execute the action represented by the agent $\langle c_j . e_j \rangle$ (recall that $c_j =_{AC1} i_p \circ \ldots \circ i_q$), we need to perform $q - p + 1$ reactions. In these reactions the first of the participating agents is one of the primitive agents $\langle \emptyset . i_p \rangle, \ldots, \langle \emptyset . i_q \rangle$, and the second one is at the beginning the agent $\langle c_j . e_j \rangle$, and in the consecutive reactions the agent which has been produced in the preceding reaction, respectively. Assuming that $\langle \emptyset . i_p \rangle$ and $\langle i_p \circ \ldots \circ i_q . e_j \rangle$ are the first two agents, we can use the first clause of (3.5)

$$\mathit{interact}(\langle C_1 . E_1 \circ D \rangle \circ \langle C_2 \circ D . E_2 \rangle \circ S',\ \langle C_1 \circ C_2 . E_1 \circ E_2 \rangle \circ S')) \leftarrow D \neq_{AC1} \emptyset.$$

to perform a reaction. Now, the substitution

$$\begin{array}{rllll} \theta_3 & = \{ & I' & \mapsto & \langle \emptyset . i_1 \rangle \circ \ldots \circ \langle \emptyset . i_p \rangle \circ \ldots \circ \langle \emptyset . i_m \rangle \circ \langle i_p \circ \ldots \circ i_q . e_j \rangle \circ ! \langle c_1 . e_1 \rangle \circ \ldots \circ ! \langle c_k . e_k \rangle, \\ & & G' & \mapsto & \langle \emptyset . g_1 \circ Z_1 \rangle \circ \ldots \circ \langle \emptyset . g_n \circ Z_n \rangle \circ Z \ \} \end{array}$$

is a unifier for the literal contained in the last goal of (3.14), and the head of a variant of the second clause of (3.4) yielding the next step in the SLDENF–derivation

$$\begin{array}{l} \leftarrow \mathit{interact}(\langle \emptyset . i_1 \rangle \circ \ldots \circ \langle \emptyset . i_p \rangle \circ \ldots \circ \langle \emptyset . i_m \rangle \circ \langle i_p \circ \ldots \circ i_q . e_j \rangle \circ ! \langle c_1 . e_1 \rangle \circ \ldots \circ ! \langle c_k . e_k \rangle, T') \wedge \\ \quad \mathit{transform}(T', \langle \emptyset . g_1 \circ Z_1 \rangle \circ \ldots \circ \langle \emptyset . g_n \circ Z_n \rangle \circ Z) \end{array} \tag{3.15}$$

Furthermore, we find a substitution

$$\begin{array}{rllll} \theta_4 & = \{ & C_1 & \mapsto & \emptyset, \\ & & E_1 & \mapsto & \emptyset, \\ & & D & \mapsto & i_p, \\ & & C_2 & \mapsto & i_{p+1} \circ \ldots \circ i_q, \\ & & E_2 & \mapsto & e_d, \\ & & S' & \mapsto & \langle \emptyset . i_1 \rangle \circ \ldots \circ \langle \emptyset . i_{p-1} \rangle \circ \langle \emptyset . i_{p+1} \rangle \circ \ldots \circ \langle \emptyset . i_m \rangle \circ ! \langle c_1 . e_1 \rangle \circ \ldots \circ ! \langle c_k . e_k \rangle, \\ & & T' & \mapsto & \langle i_{p+1} \circ \ldots \circ i_q . e_d \rangle \circ S' \ \}, \end{array}$$

which is a unifier for the first literal in (3.15) and the head of the first clause of (3.5). With the help of this clause and the substitution θ_4 we can continue the SLDENF–derivation in (3.15) by

$$\begin{array}{l} \leftarrow i_p \neq_{AC1} \emptyset \wedge \\ \quad \mathit{transform}(\langle \emptyset . i_1 \rangle \circ \ldots \circ \langle \emptyset . i_{p-1} \rangle \circ \langle \emptyset . i_{p+1} \rangle \circ \ldots \circ \langle \emptyset . i_m \rangle \circ \langle i_{p+1} \circ \ldots \circ i_q . e_d \rangle \circ ! \langle c_1 . e_1 \rangle \circ \ldots \circ ! \langle c_k . e_k \rangle, \\ \qquad \langle \emptyset . g_1 \circ Z_1 \rangle \circ \ldots \circ \langle \emptyset . g_n \circ Z_n \rangle \circ Z) \\ \leftarrow \mathit{transform}(\langle \emptyset . i_1 \rangle \circ \ldots \circ \langle \emptyset . i_{p-1} \rangle \circ \langle \emptyset . i_{p+1} \rangle \circ \ldots \circ \langle \emptyset . i_m \rangle \circ \langle i_{p+1} \circ \ldots \circ i_q . e_d \rangle \circ ! \langle c_1 . e_1 \rangle \circ \ldots \circ ! \langle c_k . e_k \rangle, \\ \qquad \langle \emptyset . g_1 \circ Z_1 \rangle \circ \ldots \circ \langle \emptyset . g_n \circ Z_n \rangle \circ Z) \end{array} \tag{3.16}$$

Obviously the literal $i_p \neq_{AC1} \emptyset$ in the first goal of (3.16) can be resolved yielding the final goal of (3.16). Note, that the agent $\langle i_{p+1} \circ \ldots \circ i_q . e_j \rangle$ produced in the reaction now represents an opportunity. Subsequent and in the same manner as demonstrated in (3.15) and (3.16), the remaining primitive agents $\langle \emptyset . i_{p+1} \rangle, \ldots, \langle \emptyset . i_q \rangle$ react until the agent $\langle \emptyset . e_j \rangle$ is produced, which obviously yields a final goal of the form:

$$\begin{aligned}\leftarrow \mathit{transform}(&\langle\emptyset.i_1\rangle \circ \ldots \circ \langle\emptyset.i_{p-1}\rangle \circ \langle\emptyset.i_{q+1}\rangle \circ \ldots \circ \langle\emptyset.i_m\rangle \circ \langle\emptyset.e_d\rangle \circ ! \langle c_1.e_1\rangle \circ \ldots \circ ! \langle c_k.e_k\rangle, \\ &\langle\emptyset.g_1 \circ Z_1\rangle \circ \ldots \circ \langle\emptyset.g_n \circ Z_n\rangle \circ Z).\end{aligned} \tag{3.17}$$

Let the term $\langle\emptyset.i_1\rangle \circ \ldots \circ \langle\emptyset.i_{p-1}\rangle \circ \langle\emptyset.i_{q+1}\rangle \circ \ldots \circ \langle\emptyset.i_m\rangle \circ \langle\emptyset.e_d\rangle \circ ! \langle c_1.e_1\rangle \circ \ldots \circ ! \langle c_k.e_k\rangle$ contained in the first argument of (3.17) represent the situation $\hat{s''}$. Obviously $\hat{s'} =_{AC1} \hat{s''}$, which shows that we have found a way to derive the first goal of (3.11) from the initial goal (3.7). The result follows immediately from appending the SLDENF–refutation in (3.11) to the concatenation of the SLDENF–derivations in (3.14), (3.15), (3.16) up to (3.17). Where (3.17) can be obtained from (3.16) by repeated use of the second clause of (3.4) and the first clause of (3.5) until (3.17) is reached. ■

One should observe that the plans generated by the two refutations are identical and, moreover, that the answer substitutions correspond in the sense that all resources occurring in the final goal situation of the SLDE–refutation occur also in the final goal situation of the SLDENF–refutation. Additionally, the latter contains the generators encoding actions.

The converse of Theorem 1 does not hold. E.g., there is no SLDE–refutation corresponding to the SLDENF–refutation in (3.6) as in the ELP–approach actions are only applicable if all of their conditions are satisfied. The major extension, however, is the treatment of actions as resources. This is also illustrated in (3.6), where the two actions are consumed during the refutation — and this cannot be modeled within ELP.

Finally, the new approach is also purely declarative. A standard semantics can easily be obtained using Clark's completion [Cla77], and it can be shown that the new approach is sound and complete wrt. this semantics. Moreover, the semantics corresponds precisely to the multiset rewritings presented earlier in this section.

3.2 Hierarchies

Before devoting our attention to the question of how to model hierarchies, we want to sum up what we have achieved so far. We have first demonstrated that by treating actions as resources we can model the tea-example and solve it in different ways including the conventional way of first changing the dollar note and then inserting the quarters for buying some tea. Since actions now are resources (called agents) they are consumed as soon as they are applied. We then have extended the model by providing more than one agent for an action. Since we sometimes do not know how often a particular action needs to be applied, in a next step we introduced generators for the controlled duplication of agents. With the help of these constructs the equational logic program developed in the previous Section 3.1.3 is capable of solving simple planning problems in a purely declarative way. In the following we will extend the approach to handle hierarchical planning as well.

3.2.1 Hierarchical Planning — an Overview

The term *hierarchical planning* is used in many different ways in the planning literature. In this section we first want to clarify what we understand by hierarchical planning; second we want to state the basic ideas of how to extend the ELP–approach to handle hierarchical planning problems.

Hierarchical planning is a method of planning in which, in order to solve a certain problem, one searches as a first attempt for a solution in a simplified representation of the problem space where unimportant details are ignored. The reason for doing so is that the "high level" problem is far easier to solve than its "low level" version one will be confronted with eventually. When a solution to the problem in the simplified representation is discovered, it remains to refine the problem space by gradually introducing more details until the final solution is found. The process of refinement can be regarded as solving a sequence of subproblems in the original problem space. Once they are solved, a solution to the overall problem is achieved. Considering this aspect of hierarchical planning, it can be regarded as a heuristic method for reducing the search space in problem solving.

Finding good abstractions is a difficult problem even for humans. In general, an abstract or simplified representation of a problem space differs from its ground space enough to achieve a significant improvement in problem-solving efficiency. At the same time the difference should not be so much as to make the mapping from abstraction space to ground space complex and time-consuming. In ABSTRIPS [Sac73] this criterion is met by making the abstraction spaces differ from their ground spaces only in the level of detail used to specify the conditions of actions.

In our extension to ELP, the generators provide a source of actions, specified on a particular level of abstraction. However, such an action might not be applicable in the real world since it is too abstract to be executed directly. It needs to be synthesized from simpler actions on a lower abstraction level. The process of synthesizing such an action can be considered as solving a subproblem in the next lower abstraction space.

The chemical metaphor our approach is based on, allows to stratify situations by encapsulating sub-situations within membranes, where certain reactions can occur locally. A membrane is porous to allow the exchange of agents between the encapsulated solution and its environment. Thus, to refine the representation of a problem space, we simply replace a generator by a membrane containing more detailed information about the problem space, i.e. agents representing some of the facts we previously abstracted from, and possibly generators for simpler actions. Local reactions performed within such a membrane synthesize a more complex agent from the encapsulated resources. The synthesized agent can then be pushed through the membrane to be used on the next higher level instead of its simpler equivalent. In this respect a membrane plays the role of a generator. The new agent might have some more effects compared with its simpler equivalent. This is, of course, no surprise since we have developed a more refined representation of the search space and, consequently, more information about the conditions and effects of the involved actions becomes available.

The process of refining the problem space can be continued further until the generators on the lowest abstraction level represent actions which are simple enough and in sufficient detail to be applicable directly for solving the given problem. Note that this method allows to refine actions by introducing different levels of abstraction for different generators; in other words, the depth of the refinement might vary for different actions. In general we aim at a problem space representation in which only the actions on the lowest level of abstraction are modeled by generators, whereas the actions on the other levels are synthesized within membranes and then via a specific mechanism pushed through the membrane to the next higher level. To summarize, we achieve a way of performing hierarchical planning in a purely declarative way.

3.2.2 An Example

Returning to the tea example, we now want to show how to stratify its problem space. Section 3.1.2 contains the derivation of the solution when using a generator for the *get–tea* action on the highest abstraction level. In the following we gradually introduce two additional abstraction levels. First we include tea bags (b) into the consideration. Since they need to be restocked once they have been used up, we do not represent them by generators, but by a certain number of agents of the form $\langle\emptyset.\{b\}\rangle$. For the moment assume that there is only one such agent representing the *last* tea bag. To this end, the generator $!\langle\{q,q,q\}.\{t\}\rangle$ will be replaced by a membrane which encapsulates the last tea bag and a new generator $!\langle\{b,q,q,q\}.\{t\}\rangle$, representing a more detailed *get–tea* action which requires a tea bag as well as the three quarters in order to be activated to produce some tea.

One should observe that we still ignore the water (w) and electricity (e) needed in reality; for simplification we assume that Bert changed the dollar note already, thus the highest level contains four quarters rather than the dollar note and the action *get–change*. Formally, each abstraction level will be enclosed in a *membrane*, which is a unary function symbol denoted by a box. Recall that each level denotes a multiset of resources, but for notational convenience we omit the multiset brackets when denoting levels. The following transformation shows how through local reactions the new agent $\langle\{q,q,q\}.\{t\}\rangle$ is synthesized from the other resources encapsulated in the inner membrane:

Initial situation:

$$\boxed{\langle\emptyset.\{q,q,q,q\}\rangle, \boxed{\langle\emptyset.\{b\}\rangle, \underline{!\langle\{b,q,q,q\}.\{t\}\rangle}}}$$

$$\Downarrow$$

$$\boxed{\langle\emptyset.\{q,q,q,q\}\rangle, \boxed{\underline{\langle\emptyset.\{b\}\rangle}, \underline{\langle\{b,q,q,q\}.\{t\}\rangle}, !\langle\{b,q,q,q\}.\{t\}\rangle}}$$

$$\Downarrow$$

$$\boxed{\langle\emptyset.\{q,q,q,q\}\rangle, \boxed{\langle\{q,q,q\}.\{t\}\rangle, !\langle\{b,q,q,q\}.\{t\}\rangle}}$$

One should observe that, since the last tea bag has been used, no other *get–tea* agent can be synthesized unless the tea bags are restocked.

In order to hand the agent $\langle\{q,q,q\}.\{t\}\rangle$ over to the next level, we introduce a new mechanism called the *airlock* [BB90]. It extracts an agent from a situation within a membrane, keeps the rest of the situation within this membrane, and isolates the extracted agent within an airlock attached to the membrane. The airlock construct is written $\langle\mathcal{C}.\mathcal{E}\rangle \lhd \boxed{\mathcal{S}}$, where $\langle\mathcal{C}.\mathcal{E}\rangle$ is the isolated agent and $\mathcal{S}$ is a situation encapsulated in a membrane containing the remaining resources. An agent in an airlock can be disconnected from the membrane it is attached to; it is then free to react with any agent in its new environment.

Using the airlock mechanism we can proceed further and perform the following transformation:

$$\begin{array}{c}
\boxed{\langle\emptyset.\{q,q,q,q\}\rangle,\ \boxed{\underline{\langle\{q,q,q\}.\{t\}\rangle},!\langle\{b,q,q,q\}.\{t\}\rangle}} \\
\Downarrow \\
\boxed{\langle\emptyset.\{q,q,q,q\}\rangle,\underline{\langle\{q,q,q\}.\{t\}\rangle} \lhd \boxed{!\langle\{b,q,q,q\}.\{t\}\rangle}} \\
\Downarrow \\
\boxed{\underline{\langle\emptyset.\{q,q,q,q\}\rangle},\underline{\langle\{q,q,q\}.\{t\}\rangle},\ \boxed{!\langle\{b,q,q,q\}.\{t\}\rangle}} \\
\Downarrow \\
\boxed{\langle\emptyset.\{t,q\}\rangle,\ \boxed{!\langle\{b,q,q,q\}.\{t\}\rangle}}
\end{array}$$

After pushing the synthesized *get–tea* agent through the membrane, it is disconnected and reacts with the quarters, producing a new agent $\langle\emptyset.\{t,q\}\rangle$ which represents some tea and the remaining quarter.

In a further refinement we intend to replace the generator $!\langle\{b,q,q,q\}.\{t\}\rangle$ by an additional membrane; this time containing a generator for the even more detailed agent $\langle\{e,w,b,q,q,q\}.\{t\}\rangle$ (again called *get–tea*), and two more generators representing a permanent supply of both water ($!\langle\emptyset.\{w\}\rangle$) and electricity ($!\langle\emptyset.\{e\}\rangle$), respectively. The transformation below demonstrates how the agent $\langle\{b,q,q,q\}.\{t\}\rangle$ is synthesized from the other resources. It can then be pushed through the membrane and the derivation proceeds as above. For simplification we only show the interactions within the new membrane.

$$\begin{array}{c}
\boxed{\underline{!\langle\emptyset.\{e\}\rangle},\ \underline{!\langle\emptyset.\{w\}\rangle},\ \underline{!\langle\{e,w,b,q,q,q\}.\{t\}\rangle}} \\
\Downarrow \\
\boxed{\langle\emptyset.\{e\}\rangle,\ \underline{\langle\emptyset.\{w\}\rangle},\ \underline{\langle\{e,w,b,q,q,q\}.\{t\}\rangle},\ !\langle\emptyset.\{e\}\rangle,\ !\langle\emptyset.\{w\}\rangle,\ !\langle\{e,w,b,q,q,q\}.\{t\}\rangle} \\
\Downarrow \\
\boxed{\underline{\langle\emptyset.\{e\}\rangle},\ \underline{\langle\{e,b,q,q,q\}.\{t\}\rangle},\ !\langle\emptyset.\{e\}\rangle,\ !\langle\emptyset.\{w\}\rangle,\ !\langle\{e,w,b,q,q,q\}.\{t\}\rangle} \\
\Downarrow \\
\boxed{\langle\{b,q,q,q\}.\{t\}\rangle,\ !\langle\emptyset.\{e\}\rangle,\ !\langle\emptyset.\{w\}\rangle,\ !\langle\{e,w,b,q,q,q\}.\{t\}\rangle}
\end{array}$$

When putting all the levels resulting out of the final refinement of the *get–tea* agent together, we obtain this initial situation:

$$\begin{array}{c}
\text{Initial situation:} \\
\boxed{\langle\emptyset.\{q,q,q,q\}\rangle,\ \boxed{\langle\emptyset.\{b\}\rangle,\ \boxed{!\langle\emptyset.\{e\}\rangle,\ !\langle\emptyset.\{w\}\rangle,\ !\langle\{e,w,b,q,q,q\}.\{t\}\rangle}}}
\end{array} \tag{3.18}$$

By performing several reactions and using the generators and the airlock mechanism, the agent representing the opportunity to get some tea for just three quarters can be synthesized in the inner membranes. After pushing it through the middle membrane, Bert can use it to get his tea.

We want to emphasize that the designer of the system is free to decide when he wants to push an agent to the next level. In the example we have applied the airlock mechanism once there were no other agents on the same level to react with such an agent, so that, by transferring it to the next level, we hoped to achieve further useful reactions. In Section 5.2, where we give an outlook on further research we suggest to introduce binary function symbols representing membranes instead of unary ones. The additional argument could then be used to specify templates which have to be matched by agents before being pushed through the membrane. This way it becomes possible to define which agents can be transferred onto the next higher level of the hierarchy.

3.2.3 The Formal Model

In Section 3.1 we have argued for representing actions as resources. There, a situation was represented by a multiset of agents and generators. To model hierarchies we additionally need membranes. Hence, a *situation* now is represented by a multiset of agents, generators, and membranes. An *agent* is a pair of multisets, where the elements of the multisets are simple terms. The first element of an agent denotes the conditions, the second element the effects of that agent. Agents with empty conditions are sometimes called *primitive agents*. These denote facts about a situation, whereas non-primitive agents denote actions. A *generator* is an expression of the form $!a$, where a is an agent. Generators denote unlimited sources of agents. A *membrane* is an expression of the form $\boxed{\mathcal{S}}$, where $\mathcal{S}$ is a situation. Membranes denote encapsulated subsituations. One should observe that the top–level situation can itself be viewed as a membrane (c.f. (3.18)).

The transition from one situation $\mathcal{S}$ to the next situation $\mathcal{S}'$ is defined by means of four interaction rules. In the following we are going to formally describe each rule giving a corresponding interaction sequence $\mathcal{S} \Rightarrow \mathcal{S}'$ in each case. Note that interaction rules are rewrite rules; they can be applied in any context matching their conditions. Thus the interaction sequences given below denote possible transitions between two situations $\mathcal{S}$ and $\mathcal{S}'$ resulting out of the application of an interaction rule.

1. A *reaction* between two agents $\langle \mathcal{C}_1.\mathcal{E}_1 \rangle$ and $\langle \mathcal{C}_2.\mathcal{E}_2 \rangle$ contained in the solution $\mathcal{S}$ with $\mathcal{E}_1 = \mathcal{E}_1' \dot\cup \mathcal{D}$, and $\mathcal{C}_2 = \mathcal{C}_2' \dot\cup \mathcal{D}$ such that $\mathcal{D} \dot\subseteq \mathcal{C}_2 \dot\cap \mathcal{E}_1$, for some $\mathcal{D} \dot{\neq} \{\}$ is defined as a transition from $\mathcal{S}$ to a situation $\mathcal{S}'$ in which these two agents are consumed, and a new agent $\langle \mathcal{C}_1 \dot\cup \mathcal{C}_2'.\mathcal{E}_2 \dot\cup \mathcal{E}_1' \rangle$ is produced. A reaction can be denoted as an interaction sequence of the following form:

$$\boxed{\ldots, \underline{\langle \mathcal{C}_1.\mathcal{E}_1' \dot\cup \mathcal{D} \rangle}, \underline{\langle \mathcal{C}_2' \dot\cup \mathcal{D}.\mathcal{E}_2 \rangle}, \ldots}$$
$$\Downarrow$$
$$\boxed{\ldots, \langle \mathcal{C}_1 \dot\cup \mathcal{C}_2'.\mathcal{E}_1' \dot\cup \mathcal{E}_2 \rangle, \ldots}$$

2. A *generation* is an interaction in which a situation $\mathcal{S}$ containing a generator $!\langle \mathcal{C}.\mathcal{E} \rangle$ is transformed into a situation $\mathcal{S}'$ containing the agent $\langle \mathcal{C}.\mathcal{E} \rangle$ in addition to the elements contained in $\mathcal{S}$. It can be denoted by an interaction sequence:

$$\boxed{\ldots, \underline{!\langle \mathcal{C}.\mathcal{E} \rangle}, \ldots}$$
$$\Downarrow$$
$$\boxed{\ldots, \langle \mathcal{C}.\mathcal{E} \rangle, !\langle \mathcal{C}.\mathcal{E} \rangle, \ldots}$$

The following two rules are rules implementing the functionality of the *airlock mechanism*. This mechanism is used to push an agent contained in a membrane through this membrane in order to make it available for interactions in the situation surrounding the membrane the agent was contained in before.

3. A situation $\mathcal{S}$ holding an encapsulated subsituation $\mathcal{S}_s$ which contains — amongst others — an agent $\langle \mathcal{C}.\mathcal{E} \rangle$ can be transformed via the *airlock* rule into a situation $\mathcal{S}'$ in which the agent $\langle \mathcal{C}.\mathcal{E} \rangle$ is extracted from $\mathcal{S}_s$, the rest $\hat{\mathcal{S}_s}$ of $\mathcal{S}_s$ remains encapsulated in a membrane, and the extracted agent $\langle \mathcal{C}.\mathcal{E} \rangle$ is attached to this membrane via an airlock, which is written $\langle \mathcal{C}.\mathcal{E} \rangle \lhd \boxed{\hat{\mathcal{S}_s}}$. The application of the airlock rule is denoted by the following sequence of interactions:

$$\boxed{\ldots, \boxed{\{\, \underline{\langle \mathcal{C}.\mathcal{E} \rangle} \,\} \,\dot{\cup}\, \hat{\mathcal{S}_s}}, \ldots}$$
$$\Downarrow$$
$$\boxed{\ldots,\ \langle \mathcal{C}.\mathcal{E} \rangle \lhd \boxed{\hat{\mathcal{S}_s}}, \ldots}$$

4. A situation $\mathcal{S}$ containing — amongst others — an agent $\langle \mathcal{C}.\mathcal{E} \rangle$ attached to a membrane $\boxed{\hat{\mathcal{S}}}$ in $\mathcal{S}$ via an airlock can be disconnected from this membrane by the *disconnect* rule. In the consecutive situation $\mathcal{S}'$ the agent $\langle \mathcal{C}.\mathcal{E} \rangle$ is free to react with any agent in its environment. The disconnection can be denoted by an interaction sequence of the form:

$$\boxed{\ldots,\ \langle \mathcal{C}.\mathcal{E} \rangle \lhd \boxed{\hat{\mathcal{S}}}, \ldots}$$
$$\Downarrow$$
$$\boxed{\ldots,\ \langle \mathcal{C}.\mathcal{E} \rangle, \boxed{\hat{\mathcal{S}}}, \ldots}$$

Returning to the representation of situations within the new approach, we can consider situations as tree–like hierarchical objects, with a (sub-)situation $\mathcal{S}$ being a vertex v labeled by the agents and generators occurring in $\mathcal{S}$, and the membranes occurring in $\mathcal{S}$ are the children of v . One should observe that the leaves of such a tree contain only agents and generators.

For example, the structure of the initial situation of the tea–scenario corresponding to (3.18) is depicted below.

Top level:

$$\langle \emptyset.\{q\} \rangle, \langle \emptyset.\{d\} \rangle, !\langle d.\{q,q,q,q\} \rangle$$
$$|$$
$$\langle \emptyset.\{b\} \rangle$$
$$|$$

Leaf:

$$!\langle \emptyset.\{e\} \rangle,\ !\langle \emptyset.\{w\} \rangle,\ !\langle \{e,w,b,q,q,q\}.\{t\} \rangle$$

Such a tree defines the hierarchical structure underlying the hierarchical planning problem. Within each vertex the usual interactions defined above may take place. The communication between a child and its father in the tree is achieved via the airlock–mechanism, which allows to transfer an agent from a subsituation, i.e. a membrane, to its surrounding situation.

A *hierarchical planning problem* consists of two situations $\mathcal{I}$ and $\mathcal{G}$, and is the question whether there exists a sequence of interactions such that $\mathcal{I}$ can be transformed into $\mathcal{G}$.

3.2.4 Equational Logic Programming and Hierarchical Planning

In this subsection we extend the equational logic program developed in Subsection 3.1.3 to be able to handle hierarchical planning problems. To this end, there is no need to extend the clauses (3.4) defining the *transform* predicate or to modify the equational theory AC1, but we have to extend (3.5), i.e. the definition of the *interact* predicate.

$$\begin{array}{l}
interact(\langle C_1.E_1 \circ D\rangle \circ \langle C_2 \circ D.E_2\rangle \circ Z,\ \langle C_1 \circ C_2.E_1 \circ E_2\rangle \circ Z) \leftarrow D \neq_{AC1} \emptyset. \\
interact(!\langle C.E\rangle \circ Z, \langle C.E\rangle \circ !\langle C.E\rangle \circ Z). \\
interact(Z \circ \boxed{S}, Z \circ \boxed{S'}) \leftarrow interact(S, S'). \\
interact(Z \circ \langle C.E\rangle \lhd \boxed{S}, Z \circ \langle C.E\rangle \lhd \boxed{S'}) \leftarrow interact(S, S'). \\
interact(Z \circ \boxed{\langle C.E\rangle \circ S}, Z \circ \langle C.E\rangle \lhd \boxed{S}). \\
interact(Z \circ \langle C.E\rangle \lhd \boxed{S}, Z \circ \langle C.E\rangle \circ \boxed{S}).
\end{array} \tag{3.19}$$

In words, the first two rules of (3.19) are just the rules from (3.5) specifying the interaction of two agents as well as the behavior of the generators. The third and forth rule of (3.19) recursively define that interactions can take place within membranes in a situation and membranes attached to an airlock, respectively. The last two rules of (3.19) define the airlock mechanism itself, i.e. constructing and disconnecting an airlock, respectively.

It is easy to see that the equational logic program consisting of the clauses (3.4) and (3.19) along with the equational theory AC1 can handle the hierarchical planning examples discussed in this work — and this holds in general. To prove such a claim we first have to represent situations in the sense of Subsection 3.2.3 in the equational logic. Let $\mathcal{S}$ be a situation. Agents and generators are represented as in Theorem 1. Membranes are represented using a unary function written as a box. For notational convenience, the top–level situation is also encapsulated into a box. For example, the initial situation (3.18) is represented by the term

$$\boxed{\langle\emptyset.q \circ q \circ q \circ q\rangle \circ \boxed{\langle\emptyset.b\rangle \circ \boxed{!\langle\emptyset.e\rangle \circ !\langle\emptyset.w\rangle \circ !\langle e \circ w \circ b \circ q \circ q \circ q.t\rangle}}} \tag{3.20}$$

while the corresponding goal situation is represented by the term

$$\boxed{\langle\emptyset.t \circ Z\rangle \circ S}.$$

In order to deal with hierarchical planning and abstraction, situations are stratified by means of membranes. Next we are going to define two important terms viz. the *depth* and the *level* of a situation. Let d be the depth of a situation $\mathcal{S}$. If $\mathcal{S}$ does not contain any subsituation then it is of depth $d = 1$. If $\mathcal{S}$ contains k ($k \geq 1$) subsituations of depth $d_1, \ldots, d_k$ then $\mathcal{S}$ is of depth $d = \max(d_1, \ldots, d_k) + 1$.

Intuitively, each membrane encapsulating a subsituation can be regarded as representing one level of abstraction. In the following the outermost or top level of a situation $\mathcal{S}$ of depth d will be referred to as level $n = 1$. A subsituation $\mathcal{S}_j$ contained in the top level situation $\mathcal{S}$ has level $n_j = d - d_j + 1$, where d denotes the depth of $\mathcal{S}$, and d_j denotes the depth of the situation $\mathcal{S}_j$, respectively. Consequently, if $\mathcal{S}_d$ denotes the subsituation on the deepest level d of $\mathcal{S}$, it will be assigned level $n_d = d$. E.g. the situation in (3.20) is of depth $d = 3$. The tea bag $\langle\emptyset.b\rangle$ is contained in level 2.

We can define two mappings $\hat{\psi}$, and $\hat{\psi}^{-1}$ similar to the mappings ψ and ψ^{-1} defined in (2.2) and (2.3), respectively. Let s be an AC1-term denoting a situation as defined in Section 3.2.3, a_i $(1 \leq i \leq n)$ be either a generator or an agent, c and e be simple AC1–terms representing the conditions and effects of an agent or generator.

$$
\begin{array}{rcl}
\hat{\psi}(\emptyset) & = & \{\!\!\}\\
\hat{\psi}(a_1 \circ \ldots \circ a_n) & = & \{\!\!\{\hat{\psi}(a_1)\}\!\!\} \,\dot{\cup}\, \hat{\psi}(a_2 \circ \ldots \circ a_n)\\
\hat{\psi}(\langle c.e\rangle) & = & \langle \psi(c).\psi(e)\rangle\\
\hat{\psi}(!\langle c.e\rangle) & = & !\langle \psi(c).\psi(e)\rangle\\
\hat{\psi}(\boxed{s}) & = & \boxed{\hat{\psi}(s)}\\
\hat{\psi}(a \lhd \boxed{s}) & = & \hat{\psi}(a) \lhd \hat{\psi}(\boxed{s})
\end{array}
\tag{3.21}
$$

The inverse function to $\hat{\psi}$ is a mapping from ground multisets into AC1–terms. It is denoted by $\hat{\psi}^{-1}$, and is defined as follows. Let $\mathcal{S}$ be a multiset denoting a situation in the new ELP, a_i $(1 \leq i \leq n)$ be either a generator or an agent, $\mathcal{C}$ and $\mathcal{E}$ be multisets of simple terms.

$$
\begin{array}{rcl}
\hat{\psi}^{-1}(\{\!\!\}) & = & \emptyset\\
\hat{\psi}^{-1}(\{\!\!\{a_1, \ldots, a_n\}\!\!\}) & = & \hat{\psi}^{-1}(a_1) \circ \hat{\psi}^{-1}(\{\!\!\{a_2, \ldots a_n\}\!\!\})\\
\hat{\psi}^{-1}(\langle \mathcal{C}.\mathcal{E}\rangle) & = & \langle \psi^{-1}(\mathcal{C}).\psi^{-1}(\mathcal{E})\rangle\\
\hat{\psi}^{-1}(!\langle \mathcal{C}.\mathcal{E}\rangle) & = & !\langle \psi^{-1}(\mathcal{C}).\psi^{-1}(\mathcal{E})\rangle\\
\hat{\psi}^{-1}(\boxed{\mathcal{S}}) & = & \boxed{\hat{\psi}^{-1}(\mathcal{S})}\\
\hat{\psi}^{-1}(a \lhd \boxed{\mathcal{S}}) & = & \hat{\psi}^{-1}(a) \lhd \hat{\psi}^{-1}(\boxed{\mathcal{S}})
\end{array}
\tag{3.22}
$$

Again, one should observe that $\hat{\psi}^{-1}(\hat{\psi}(s)) =_{AC1} s$ and $\hat{\psi}(\hat{\psi}^{-1}(\mathcal{S})) \doteq \mathcal{S}$, where s is an AC1–term and $\mathcal{S}$ is a multiset denoting a situation. Furthermore, $\psi(s) \doteq \mathcal{S}$ iff $s =_{AC1} \psi^{-1}(\mathcal{S})$.

Having specified the equational logic program consisting of the clauses in (3.4) and (3.19) together with the equational theory AC1, and the mappings between multisets and AC1–terms in the extended representation (see (3.21) and (3.22)), we now want to prove that the extended ELP can handle the hierarchical planning problems addressed in this work (see the end of Section 3.2.3). In order to do so we first need to prove the following lemma.

Lemma 2 *Let $\mathcal{S}_1$ and $\mathcal{S}_2$ be multisets representing two situations such that $\mathcal{S}_2$ was obtained from $\mathcal{S}_1$ by applying one of the interaction rules specified in Section 3.2.3 to $\mathcal{S}_1$. Furthermore, let $s_1 = \hat{\psi}^{-1}(\mathcal{S}_1)$ and $s_2 = \hat{\psi}^{-1}(\mathcal{S}_2)$ be the AC1–terms representing $\mathcal{S}_1$ and $\mathcal{S}_2$, respectively. There exists an interaction transforming $\mathcal{S}_1$ into $\mathcal{S}_2$ iff there exists an SLDENF–refutation for $\{\leftarrow \mathrm{interact}(s_1, s_2)\} \cup \{(3.19)\}$.*

Proof

First we want to prove the *only-if-half* of the lemma. The proof is by induction on the level n of the situations $\mathcal{S}_1, \mathcal{S}_2$ on which the interaction rule was applied. Let

$$\mathcal{S}_1 \stackrel{R_1}{\Longrightarrow} \mathcal{S}_2$$

denote an interaction from $\mathcal{S}_1$ into $\mathcal{S}_2$ using the interaction rule R on the outermost level ($n = 1$) of $\mathcal{S}_1$. We distinguish the following four cases wrt. R.

1. If R was a *reaction* then $\mathcal{S}_1$ contains two agents $\langle \mathcal{C}_1.\mathcal{E}_1 \,\dot{\cup}\, \mathcal{D}\rangle$ and $\langle \mathcal{C}_2 \,\dot{\cup}\, \mathcal{D}.\mathcal{E}_2\rangle$ for some $\mathcal{D} \dot{\neq} \{\!\}$ on its outermost level, and $\mathcal{S}_2$ contains the result of the reaction on its outermost level, which is the agent $\langle \mathcal{C}_1 \,\dot{\cup}\, \mathcal{C}_2.\mathcal{E}_1 \dot{\cup}\, \mathcal{E}_2\rangle$, i.e.
$$\mathcal{S}_2 = \mathcal{S}_1 \,\dot{-}\, \{\!\!\{\langle \mathcal{C}_1.\mathcal{E}_1 \,\dot{\cup}\, \mathcal{D}\rangle, \langle \mathcal{C}_2 \,\dot{\cup}\, \mathcal{D}.\mathcal{E}_2\rangle\}\!\!\} \,\dot{\cup}\, \{\!\!\{\langle \mathcal{C}_1 \,\dot{\cup}\, \mathcal{C}_2.\mathcal{E}_1 \dot{\cup}\, \mathcal{E}_2\rangle\}\!\!\}.$$
Let
$$\begin{array}{rlcl} \theta & = \{ \; C_1 & \mapsto & \psi^{-1}(\mathcal{C}_1), \\ & E_1 & \mapsto & \psi^{-1}(\mathcal{E}_1), \\ & D & \mapsto & \psi^{-1}(\mathcal{D}), \\ & C_2 & \mapsto & \psi^{-1}(\mathcal{C}_2), \\ & E_2 & \mapsto & \psi^{-1}(\mathcal{E}_2), \\ & Z & \mapsto & \hat{\psi}^{-1}(\mathcal{S}_1 \,\dot{-}\, \{\!\!\{\langle \mathcal{C}_1.\mathcal{E}_1 \,\dot{\cup}\, \mathcal{D}\rangle, \langle \mathcal{C}_2 \,\dot{\cup}\, \mathcal{D}.\mathcal{E}_2\rangle\}\!\!\})\}. \end{array}$$
Obviously, θ is an AC1–unifier for s_1 and $\langle C_1.E_1 \circ D\rangle \circ \langle C_2 \circ D.E_2\rangle \circ Z$, and s_2 and $\langle C_1 \circ C_2.E_1 \circ E_2\rangle \circ Z$. Consequently, the goal $\leftarrow interact(s_1, s_2)$ can be resolved using θ with the first clause
$$interact(\langle C_1.E_1 \circ D\rangle \circ \langle C_2 \circ D.E_2\rangle \circ Z,\ \langle C_1 \circ C_2.E_1 \circ E_2\rangle \circ Z) \leftarrow D \neq_{AC1} \emptyset.$$
of (3.19). The resulting goal $\leftarrow D \neq_{AC1} \emptyset$ can obviously be resolved since the application of the reaction rule implicitly requires $\mathcal{D}$ to be non-empty, and thus the derivation succeeds.

2. If R was a *generation* then $\mathcal{S}_1$ contains a generator $!\langle \mathcal{C}.\mathcal{E}\rangle$ on its outermost level, and $\mathcal{S}_2 = \mathcal{S}_1 \,\dot{\cup}\, \{\!\!\{\langle \mathcal{C}.\mathcal{E}\rangle\}\!\!\}$. Let
$$\begin{array}{rlcl} \theta & = \{ \; C & \mapsto & \psi^{-1}(\mathcal{C}), \\ & E & \mapsto & \psi^{-1}(\mathcal{E}), \\ & Z & \mapsto & \hat{\psi}^{-1}(\mathcal{S}_1 \,\dot{-}\, \{\!\!\{!\langle \mathcal{C}.\mathcal{E}\rangle\}\!\!\})\}. \end{array}$$
Obviously, θ is an AC1–unifier for s_1 and $!\langle C.E\rangle \circ Z$, and s_2 and $\langle C.E\rangle \circ !\langle C.E\rangle \circ Z$. Consequently, the goal $\leftarrow interact(s_1, s_2)$ can be resolved using θ with the second clause
$$interact(!\langle C.E\rangle \circ Z, \langle C.E\rangle \circ !\langle C.E\rangle \circ Z).$$
of (3.19) yielding the empty clause immediately.

3. If R was the *airlock* rule, then $\mathcal{S}_1$ contains a subsituation of the form $\boxed{\{\!\!\{\langle \mathcal{C}.\mathcal{E}\rangle\}\!\!\} \,\dot{\cup}\, \mathcal{S}}$ on its outermost level. The successor situation $\mathcal{S}_2$ contains an airlock molecule of the form $\langle \mathcal{C}.\mathcal{E}\rangle \lhd \boxed{\mathcal{S}}$.
Let
$$\begin{array}{rlcl} \theta & = \{ \; C & \mapsto & \psi^{-1}(\mathcal{C}), \\ & E & \mapsto & \psi^{-1}(\mathcal{E}), \\ & S & \mapsto & \hat{\psi}^{-1}(\mathcal{S}), \\ & Z & \mapsto & \hat{\psi}^{-1}(\mathcal{S}_1 \,\dot{-} \{\!\!\{ \boxed{\{\!\!\{\langle \mathcal{C}.\mathcal{E}\rangle\}\!\!\} \,\dot{\cup}\, \mathcal{S}} \}\!\!\})\}. \end{array}$$
Obviously, θ is an AC1–unifier for s_1 and $Z \circ \boxed{\langle C.E\rangle \circ S}$, and s_2 and $Z \circ \langle C.E\rangle \lhd \boxed{S}$. Consequently, the goal $\leftarrow interact(s_1, s_2)$ can be resolved using θ with the fifth clause
$$interact(Z \circ \boxed{\langle C.E\rangle \circ S}, Z \circ \langle C.E\rangle \lhd \boxed{S}).$$
of (3.19) yielding the empty clause immediately.

4. If R was the *disconnection* rule, then $\mathcal{S}_1$ contains an airlock molecule of the form $\langle \mathcal{C}.\mathcal{E}\rangle \lhd \boxed{\mathcal{S}}$ on its outermost level. The subsequent situation $\mathcal{S}_2$ is of the form $\mathcal{S}_1 \dot{-} \{\!\{\langle \mathcal{C}.\mathcal{E}\rangle \lhd \boxed{\mathcal{S}}\}\!\} \dot{\cup} \{\!\{\langle \mathcal{C}.\mathcal{E}\rangle, \boxed{\mathcal{S}}\}\!\}$. Let

$$\begin{array}{rlll} \theta & = \{ & C & \mapsto \psi^{-1}(\mathcal{C}), \\ & & E & \mapsto \psi^{-1}(\mathcal{E}), \\ & & S & \mapsto \psi^{-1}(\mathcal{S}), \\ & & Z & \mapsto \hat{\psi}^{-1}(\mathcal{S}_1 \dot{-} \{\!\{\langle \mathcal{C}.\mathcal{E}\rangle \lhd \boxed{\mathcal{S}}\}\!\})\}. \end{array}$$

Obviously, θ is an AC1–unifier for s_1 and $Z \circ \langle C.E\rangle \lhd \boxed{S}$, and s_2 and $Z \circ \langle C.E\rangle \circ \boxed{S}$. Consequently, the goal $\leftarrow interact(s_1, s_2)$ can be resolved using the last clause

$$interact(Z \circ \langle C.E\rangle \lhd \boxed{S}, Z \circ \langle C.E\rangle \circ \boxed{S}).$$

of (3.19) yielding the empty clause immediately.

Suppose now that the result holds for all interaction sequences in which an interaction rule was applied on a level less than n. Let

$$\mathcal{S}_1 \xRightarrow{R_n} \mathcal{S}_2$$

denote an interaction from $\mathcal{S}_1$ into $\mathcal{S}_2$ using the interaction rule R on the nth level ($1 < n \leq d$) of $\mathcal{S}_1$, where d denotes the depth of $\mathcal{S}_1$. There are the following two cases to be distinguished.

1. $\mathcal{S}_1$ is of the form $\mathcal{Z} \dot{\cup} \boxed{\mathcal{S}}$, and the interaction is performed within $\mathcal{S}$ on the level $n-1$. The resulting situation $\mathcal{S}_2$ is then of the form $\mathcal{Z} \dot{\cup} \boxed{\mathcal{S}'}$, where $\mathcal{S}'$ denotes the result of applying the interaction rule R to $\mathcal{S}$ on the level $n-1$. Let

$$\begin{array}{rlll} \theta & = \{ & S & \mapsto \hat{\psi}^{-1}(\mathcal{S}), \\ & & S' & \mapsto \hat{\psi}^{-1}(\mathcal{S}'), \\ & & Z & \mapsto \hat{\psi}^{-1}(\mathcal{S}_1 \dot{-} \{\!\{\boxed{\mathcal{S}}\}\!\})\}, \end{array}$$

which is an AC1–unifier for s_1, s_2 and the head of the third clause

$$interact(Z \circ \boxed{S}, Z \circ \boxed{S'}) \leftarrow interact(S, S').$$

of (3.19). Applying the induction hypothesis to the transformation from $\mathcal{S}$ into $\mathcal{S}'$ (which is performed on the level $n-1$) yields the SLDENF–refutation of the resulting goal $\leftarrow interact(S\theta, S'\theta)$, and the result follows immediately.

2. $\mathcal{S}_1$ is of the form $\mathcal{Z} \dot{\cup} \{\!\{\langle \mathcal{C}.\mathcal{E}\rangle \lhd \boxed{\mathcal{S}}\}\!\}$, and the interaction is performed within $\boxed{\mathcal{S}}$ on the level $n-1$. The resulting situation $\mathcal{S}_2$ is then of the form $\mathcal{Z} \dot{\cup} \{\!\{\langle \mathcal{C}.\mathcal{E}\rangle \lhd \boxed{\mathcal{S}'}\}\!\}$, where $\mathcal{S}'$ denotes the result of applying the interaction rule R to $\mathcal{S}$ on the level $n-1$. Let

$$\begin{array}{rlll} \theta & = \{ & C & \mapsto \psi^{-1}(\mathcal{C}), \\ & & E & \mapsto \psi^{-1}(\mathcal{E}), \\ & & S & \mapsto \hat{\psi}^{-1}(\mathcal{S}), \\ & & S' & \mapsto \hat{\psi}^{-1}(\mathcal{S}'), \\ & & Z & \mapsto \hat{\psi}^{-1}(\mathcal{S}_1 \dot{-} \{\!\{\langle \mathcal{C}.\mathcal{E}\rangle \lhd \boxed{\mathcal{S}}\}\!\})\}, \end{array}$$

which is an AC1–unifier for s_1, s_2 and the head of the forth clause

$$interact(Z \circ \langle C.E\rangle \lhd \boxed{S}, Z \circ \langle C.E\rangle \lhd \boxed{S'}) \leftarrow interact(S, S').$$

of (3.19). Applying the induction hypothesis to the transformation from $\mathcal{S}$ into $\mathcal{S}'$ (which is performed on the level $n-1$) yields the SLDENF–refutation of the resulting goal $\leftarrow interact(S\theta, S'\theta)$, and the result follows immediately.

Second we want to prove the *if-half* of the theorem. Suppose the result holds for all SLDENF–refutations of length less than l. Let

$$\leftarrow interact(s_1, s_2) \xrightarrow{l}_{\text{SLDENF}} \square$$

be an SLDENF–refutation of length l. Moreover, let P be the program clause and λ be the unifier used in the first step of the above refutation. Finally, let G be the resolvent of the initial goal and P using λ, i.e.

$$\leftarrow interact(s_1, s_2) \xrightarrow{1}_{\text{SLDENF}} G \xrightarrow{l-1}_{\text{SLDENF}} \square$$

We distinguish the following six cases wrt. P.

1. If $P \equiv interact(\langle C_1.E_1 \circ D\rangle \circ \langle C_2 \circ D.E_2\rangle \circ Z,\ \langle C_1 \circ C_2.E_1 \circ E_2\rangle \circ Z) \leftarrow D \neq_{AC1} \emptyset$ then $G \equiv \leftarrow D\lambda \neq_{AC1} \emptyset$ and $l = 2$. Furthermore, we find

$$\langle C_1\lambda.E_1\lambda \circ D\lambda\rangle \circ \langle C_2\lambda \circ D\lambda.E_2\lambda\rangle \circ Z\lambda \ =_{AC1}\ s_1\lambda$$

and

$$\langle C_1\lambda \circ C_2\lambda.E_1\lambda \circ E_2\lambda\rangle \circ Z\lambda \ =_{AC1}\ s_2\lambda.$$

Hence,

$$\hat{\psi}(s_1\lambda) = \{\!\{\langle \psi(C_1\lambda).\psi(E_1\lambda) \ \dot\cup\ \psi(D\lambda)\rangle, \langle \psi(C_2\lambda) \ \dot\cup\ \psi(D\lambda).\psi(E_2\lambda)\rangle\}\!\} \ \dot\cup\ \hat{\psi}(Z\lambda) \tag{3.23}$$

and

$$\hat{\psi}(s_2\lambda) = \{\!\{\langle \psi(C_1\lambda) \ \dot\cup\ \psi(C_2\lambda).\psi(E_1\lambda) \ \dot\cup\ \psi(E_2\lambda)\rangle\}\!\} \ \dot\cup\ \hat{\psi}(Z\lambda). \tag{3.24}$$

Now, obviously (3.23) $\Rightarrow$ (3.24) with the *reaction* rule. Recall that the reaction rule implicitly requires a non-empty intersection $\psi(D\lambda)$ between the two multisets $\psi(E_1\lambda) \ \dot\cup\ \psi(D\lambda)$ and $\psi(C_2\lambda) \ \dot\cup\ \psi(D\lambda)$, thus a check whether $D\lambda \neq_{AC1} \emptyset$ is implicitly contained in the application of the reaction rule, which concludes this case.

2. If $P \equiv interact(!\langle C.E\rangle \circ Z, \langle C.E\rangle \circ !\langle C.E\rangle \circ Z)$ then G is the empty clause and $l = 1$. Furthermore, we find λ such that

$$!\langle C\lambda.E\lambda\rangle \circ Z\lambda \ =_{AC1}\ s_1\lambda$$

and

$$\langle C\lambda.E\lambda\rangle \circ !\langle C\lambda.E\lambda\rangle \circ Z\lambda \ =_{AC1}\ s_2\lambda$$

Hence,

$$\hat{\psi}(s_1\lambda) = \{\!\{!\langle \psi(C\lambda).\psi(E\lambda)\rangle\}\!\} \ \dot\cup\ \hat{\psi}(Z\lambda) \tag{3.25}$$

and

$$\hat{\psi}(s_2\lambda) = \{\!\{\langle \psi(C\lambda).\psi(E\lambda)\rangle, !\langle \psi(C\lambda).\psi(E\lambda)\rangle\}\!\} \ \dot\cup\ \hat{\psi}(Z\lambda) \tag{3.26}$$

Now, obviously, (3.25) $\Rightarrow$ (3.26) with the *generator* rule, which concludes this case.

3. If $P \equiv interact(Z \circ \boxed{\langle C.E\rangle \circ S}, Z \circ \langle C.E\rangle \lhd \boxed{S})$ then G is the empty clause and $l = 1$. Furthermore, we find

$$Z\lambda \circ \boxed{\langle C\lambda.E\lambda\rangle \circ S\lambda} =_{AC1} s_1\lambda$$

and

$$Z\lambda \circ \langle C\lambda.E\lambda\rangle \lhd \boxed{S\lambda} =_{AC1} s_2\lambda.$$

Hence,

$$\hat{\psi}(s_1\lambda) = \hat{\psi}(Z\lambda) \mathbin{\dot{\cup}} \{\!\!\{ \boxed{\{\!\!\{\langle\psi(C\lambda).\psi(E\lambda)\rangle\}\!\!\} \mathbin{\dot{\cup}} \hat{\psi}(S\lambda)} \}\!\!\} \tag{3.27}$$

and

$$\hat{\psi}(s_2\lambda) = \hat{\psi}(Z\lambda) \mathbin{\dot{\cup}} \{\!\!\{ \langle\psi(C\lambda).\psi(E\lambda)\rangle \lhd \boxed{\hat{\psi}(S\lambda)} \}\!\!\}. \tag{3.28}$$

Now, obviously, (3.27) $\Rightarrow$ (3.28) with the *airlock* rule, which concludes this case.

4. If $P \equiv interact(Z \circ \langle C.E\rangle \lhd \boxed{S}, Z \circ \langle C.E\rangle \circ \boxed{S})$ then G is the empty clause and $l = 1$. Furthermore, we find

$$Z\lambda \circ \langle C\lambda.E\lambda\rangle \lhd \boxed{S\lambda} =_{AC1} s_1\lambda$$

and

$$Z\lambda \circ \langle C\lambda.E\lambda\rangle \circ \boxed{S\lambda} =_{AC1} s_2\lambda$$

Hence,

$$\hat{\psi}(s_1\lambda) = \hat{\psi}(Z\lambda) \mathbin{\dot{\cup}} \{\!\!\{ \langle\psi(C\lambda).\psi(E\lambda)\rangle \lhd \boxed{\hat{\psi}(S\lambda)} \}\!\!\} \tag{3.29}$$

and

$$\hat{\psi}(s_2\lambda) = \hat{\psi}(Z\lambda) \mathbin{\dot{\cup}} \{\!\!\{ \langle\psi(C\lambda).\psi(E\lambda)\rangle \}\!\!\} \mathbin{\dot{\cup}} \hat{\psi}(S\lambda). \tag{3.30}$$

Now, obviously, (3.29) $\Rightarrow$ (3.30) with the *disconnect* rule, which concludes this case.

5. If $P \equiv interact(Z \circ \boxed{S_1'}, Z \circ \boxed{S_2'}) \leftarrow interact(S_1', S_2')$ then $G \equiv \leftarrow interact(S_1'\lambda, S_2'\lambda)$. Furthermore, we find

$$Z\lambda \circ \boxed{S_1'\lambda} =_{AC1} s_1\lambda$$

and

$$Z\lambda \circ \boxed{S_2'\lambda} =_{AC1} s_2\lambda.$$

Hence,

$$\hat{\psi}(s_1\lambda) = \hat{\psi}(Z\lambda) \mathbin{\dot{\cup}} \{\!\!\{ \boxed{\hat{\psi}(S_1'\lambda)} \}\!\!\} \tag{3.31}$$

and

$$\hat{\psi}(s_2\lambda) = \hat{\psi}(Z\lambda) \mathbin{\dot{\cup}} \{\!\!\{ \boxed{\hat{\psi}(S_2'\lambda)} \}\!\!\}. \tag{3.32}$$

By an application of the induction hypothesis to the refutation of $\{G\} \cup \{(3.19)\}$ we find a sequence of interactions from $\hat{\psi}(S_1'\lambda)$ into $\hat{\psi}(S_2'\lambda)$. Since the interaction rules are rewrite rules which can be applied in any context, and $\hat{\psi}(S_1'\lambda)$ is a subsituation of $\hat{\psi}(s_1\lambda)$ following (3.31), and $\hat{\psi}(S_2'\lambda)$ is a subsituation of $\hat{\psi}(s_2)$ following (3.32), we conclude that there is a sequence of interaction rules transforming (3.31) into (3.32), which concludes this case.

6. If $P \equiv interact(Z \circ \langle C.E \rangle \lhd \boxed{S_1'}, Z \circ \langle C.E \rangle \lhd \boxed{S_2'}) \leftarrow interact(S_1', S_2')$ then $G \equiv \leftarrow interact(S_1'\lambda, S_2'\lambda)$. Furthermore, we find

$$Z\lambda \circ \langle C\lambda.E\lambda \rangle \lhd \boxed{S_1'\lambda} =_{AC1} s_1\lambda$$

and

$$Z\lambda \circ \langle C\lambda.E\lambda \rangle \lhd \boxed{S_2'\lambda} =_{AC1} s_2\lambda.$$

Hence,

$$\hat{\psi}(s_1\lambda) = \hat{\psi}(Z\lambda) \dot{\cup} \{\!\{ \langle \psi(C\lambda).\psi(E\lambda) \rangle \lhd \boxed{\hat{\psi}(S_1'\lambda)} \}\!\} \tag{3.33}$$

and

$$\hat{\psi}(s_2\lambda) = \hat{\psi}(Z\lambda) \dot{\cup} \{\!\{ \langle \psi(C\lambda).\psi(E\lambda) \rangle \lhd \boxed{\hat{\psi}(S_2'\lambda)} \}\!\}. \tag{3.34}$$

By an application of the induction hypothesis to the refutation of $\{G\} \cup \{(3.19)\}$ we find a sequence of interactions from $\hat{\psi}(S_1'\lambda)$ into $\hat{\psi}(S_2'\lambda)$. Since the interaction rules are rewrite rules which can be applied in any context, and $\hat{\psi}(S_1'\lambda)$ is a subsituation with an airlock contained in $\hat{\psi}(s_1\lambda)$ following (3.33), and $\hat{\psi}(S_2'\lambda)$ is a subsituation with an airlock contained in $\hat{\psi}(s_2)$ following (3.34), we conclude that there is a sequence of interaction rules transforming (3.33) into (3.34), which concludes this case. ■

Lemma 2 proves that the extended ELP covers changes within a situation that result out of the use of a single interaction rule. However, the extended ELP is capable of handling complete interaction sequences solving hierarchical planning problems. This will be established by the following theorem.

Theorem 3 *Let $\mathcal{P}$ be a hierarchical planning problem consisting of the initial situation $\mathcal{I}$ and the goal situation $\mathcal{G}$, and let $i = \hat{\psi}^{-1}(\mathcal{I})$ and $g = \hat{\psi}^{-1}(\mathcal{G})$ be the terms representing $\mathcal{I}$ and $\mathcal{G}$, respectively. There exists a solution to $\mathcal{P}$ iff there exists an SLDENF–refutation for $\{\leftarrow transform(i, g)\} \cup \{(3.4), (3.19)\}$ wrt. AC1.*

The two implications can be proved by transfinite induction on the length l of the interaction sequence which transforms $\mathcal{I}$ into $\mathcal{G}$, and on the length l' of the SLDENF–refutation, respectively. In particular, the proof shows that the interaction sequences are identical if we neglect the applications of the third and forth rule of (3.19).

Proof
First we want to prove the *only–if*-half of the theorem. Suppose the result holds for all interaction sequences of length less than l. Let $\mathcal{I} \overset{l}{\Rightarrow} \mathcal{G}$ be an interaction sequence of length l. We distinguish the following two cases;

1. If $\mathcal{I} \doteq \mathcal{G}$ (i.e. the initial situation already establishes the goal situation), then $l = 0$ and the corresponding interaction sequence is empty. It follows that $i =_{AC1} g$, and thus the empty substitution ε is an AC1–unifier for i and g such that the goal $\leftarrow transform(i, g)$ succeeds immediately with the first clause of (3.4), which concludes this case.

2. If $\mathcal{I} \not\doteq \mathcal{G}$, then let $\mathcal{S}$ be the situation obtained by applying the interaction rule R to $\mathcal{I}$, i.e.

   $$\mathcal{I} \stackrel{1}{\Rightarrow} \mathcal{S} \stackrel{l-1}{\Rightarrow} \mathcal{G},$$

   and let $s = \hat{\psi}^{-1}(\mathcal{S})$ be the AC1–term representing $\mathcal{S}$. Let $\theta = \{I \mapsto i, G \mapsto g\}$. Using θ and the second clause

   $$\mathit{transform}(I, G) \leftarrow \mathit{interact}(I, S) \wedge \mathit{transform}(S, G).$$

   of (3.4) we can employ SLDE–resolution yielding the new goal $\leftarrow \mathit{interact}(i, S) \wedge \mathit{transform}(S, g)$. It is left to show that this goal has an SLDENF–refutation.

   From Lemma 2 we know that there is an SLDENF–refutation of $\{\leftarrow \mathit{interact}(i, S)\} \cup \{(3.19)\}$, which produces the substitution $\theta' = \{S \mapsto s\}$. Applying the induction hypothesis to the interaction sequence from $\mathcal{S}$ to $\mathcal{G}$ we find an SLDENF–refutation of the goal $\leftarrow \mathit{transform}(S\theta', g)$, and the result follows immediately as an SLDENF–refutation of the form.

   $$\begin{array}{l} \leftarrow \mathit{transform}(i, g) \\ \leftarrow \mathit{interact}(i, S) \wedge \mathit{transform}(S, g) \\ \leftarrow \mathit{transform}(s, g) \\ \vdots \\ \square \end{array}$$

Second we want to prove the *if*-half of the theorem. Suppose the result holds for all SLDENF–refutations of length less than l' . Let

$$\leftarrow \mathit{transform}(i, g) \xrightarrow{l'}_{\text{SLDENF}} \square$$

be an SLDENF–refutation of length l' . Furthermore, let P be the program clause and θ be the unifier used in the first step of this refutation. Finally, let R be the resolvent of the initial goal and P using θ , i.e.

$$\leftarrow \mathit{transform}(i, g) \xrightarrow{1}_{\text{SLDENF}} R \xrightarrow{l'-1}_{\text{SLDENF}} \square$$

We distinguish two cases wrt. P.

1. If $P \equiv \mathit{transform}(I, I)$, then R is the empty clause and $l' = 1$. Hence, we found an AC1–unifier, viz. ε such that $i =_{AC1} g$. It follows that $\mathcal{I} \doteq \mathcal{G}$, and consequently the corresponding interaction sequence is empty, since the initial situation already establishes the goal situation.

2. If $P \equiv \mathit{transform}(I, G) \leftarrow \mathit{interact}(I, S) \wedge \mathit{transform}(S, G)$, then $\theta = \{I \mapsto i, G \mapsto g\}$, and $R\theta \equiv \leftarrow \mathit{interact}(i, S) \wedge \mathit{transform}(S, g)$. Suppose that the refutation of the subgoal $\leftarrow \mathit{interact}(i, S)$ yields an answer substitution $\theta' = \{S \mapsto s\}$ such that s is an AC1–term representing the subsequent situation when applying an interaction rule to i , and $\mathcal{S} = \hat{\psi}(s)$. From Lemma 2 we know that there is a corresponding interaction sequence $\mathcal{I} \Rightarrow \mathcal{S}$. Applying the induction hypothesis to the refutation of $\leftarrow \mathit{transform}(S\theta', g)$ yields the interaction sequence $\mathcal{S} \stackrel{*}{\Rightarrow} \mathcal{G}$. The result follows immediately from the concatenation of the two interaction sequences yielding the following sequence

   $$\mathcal{I} \stackrel{1}{\Rightarrow} \mathcal{S} \stackrel{*}{\Rightarrow} \mathcal{G},$$

   which establishes that there is an interaction sequence from $\mathcal{I}$ into $\mathcal{G}$. ■

With the proof of Theorem 3 we have shown that the extension of ELP we developed within this chapter is suitable for solving the specified sort of hierarchical planning problems. The reification of actions together with the chemical metaphor, in particular, the membrane concept form an essential component of the extended ELP–approach. As demonstrated SLDENF–resolution is employed in order to construct a proof for the solution of a given planning problem. In the following section we want to discuss how a plan can be extracted from such a proof.

3.3 Plans

In general we understand (deductive) planning as a two–stage process. First we search for evidence of the solvability of the problem, in other words, we are looking for a way of transforming the initial situation into the goal situation. Second, once a proof has been established, a concrete plan shall be extracted. There are different ways of doing so. In this section we discuss some of them, considering their applicability when used in the new approach.

First let us examine some techniques for obtaining a plan as an answer substitution of an SLDENF–refutation. We could, for instance, introduce an extra argument in the predicate *transform* which, at the end of the refutation, has an answer substitution corresponding to the generated plan. This would correspond to the generation of plans in the original ELP–approach as discussed in Section 2.3.2. In the new approach a plan would be a sequence of agents taking part in reactions. Doing this, however, would force us to introduce a third argument in the predicate *interact* as well. Since the reactions are performed solely by the first rule of (3.19), the additional argument would be instantiated whenever this clause is used. The disadvantage of such a solution becomes obvious when we are going to extend the approach allowing several reactions to be performed concurrently (see Section 5.1). The need to update the additional argument forces us to install a central controller which is known to slow down any parallel computation.

An alternative is to represent agents by triples of the form $\langle \mathcal{C}.a.\mathcal{E} \rangle$, where a is the name of the agent, i.e. the name of an action. When a reaction is performed, the name of the new agent is composed from the respective names of the two agents participating in the reaction, e.g., $a_1 : a_2$ represents an agent which was produced by executing a_1 followed by a_2. In the end, each agent in the solution has a name corresponding to how it was obtained. The plan itself can then be extracted from the final situation. This alternative is very similar to the concept of fluents used within the situation calculus. Consequently, it has the same disadvantages as the situation calculus [HT95b], i.e. each agent has to be updated whenever two agents react. On the other hand it becomes difficult to represent those primitive agents already contained in the initial situation. They needed to be assigned a name from which follows that there was no action necessary to obtain them. Furthermore, this alternative also has disadvantages when considering a mapping of our extended ELP onto a modified version of the Chemical Abstract Machine (see Section 4.3). The problem is to find an adequate representation for the name of the agent when heating the agent and thereby splitting it into ions which can perform reactions.

Another way to obtain the plan as an answer substitution of the refutation is to introduce a special resource which plays the role of a state literal in the linear connection method [Bib86]. There, state literals are used to record the sequence of actions taken in order to achieve a goal (see Section 2.3.2). In this alternative each situation as well as the conditions and effects of actions would contain a state literal.

Hence, an agent would then have the form $\langle \mathcal{C} \mathbin{\dot{\cup}} \{s(P)\}.\mathcal{E} \mathbin{\dot{\cup}} \{s(a(P))\}\rangle$, where a denotes the name and $\mathcal{C}$ and $\mathcal{E}$ denote the conditions and effects of that agent. Consequently, the first rule in (3.19) would need to be modified such that all reactions have to consume the corresponding state literals or, formally, D must then contain a state literal. This way we obtain a sequential processing which corresponds to the central controller mentioned before, and thus has similar disadvantages.

The final alternative discussed here and the alternative implemented in the new approach presented in this work is to extract the plan from the SLDENF–refutation itself by means of the interactions performed during a transformation. For example, in the SLDENF–refutation (3.6) the first reaction is performed with the agent $\langle q \circ q \circ q.t\rangle$ representing the action *get–tea* achieving the agent $\langle q \circ q.t\rangle$ representing the opportunity to get some tea for only two quarters. Next the agent $\langle d.q \circ q \circ q \circ q\rangle$ representing the action *get–change* is consumed; and finally the opportunity $\langle q \circ q.t\rangle$ is consumed producing the agent $\langle \emptyset.t \circ q \circ q\rangle$. Consequently, the plan consists of starting the action *get–tea*, stopping in order to perform the action *get–change* and continuing the action *get–tea* afterwards.

Methods like this are widely applied in the fields of program synthesis and constructive logic. For our purposes it has the following advantages. First, the entire transformation process becomes shorter and appears to be easier to illustrate. Second, this method comes very close to the chemical metaphor, whereon the approach is based. It is therefore easier to find an adequate abstract machine model. Last, it turned out to be very similar to the semantics of the $\multimap$ operator in linear logic [Gir87] in so far as an agent $\langle \mathcal{C}.\mathcal{E}\rangle$ can be interpreted as the material implication $\mathcal{C} \multimap \mathcal{E}$ between its conditions $\mathcal{C}$ and its effects $\mathcal{E}$.

Let us now summarize what we have achieved so far. Within this chapter the equational logic programming approach has been further extended to handle hierarchical planning problems. Hierarchies are an essential ingredient for eventually turning the deductive reasoning system into a full–fledged planning system. The central idea presented is that in a hierarchical planning system the actions itself are resources. They are consumed as soon as they are applied, and they are produced by either generators or reactions with other resources. Furthermore, the membrane notion is an important extension to the ELP–approach. It allows to stratify a planning problem into different levels of abstraction, such that within a membrane complex actions can be synthesized from less complex resources. Using the airlock mechanism these complex actions can be pushed through the membrane onto the next higher level of abstraction where they can perform further reactions. Thus, hierarchical planning becomes possible.

In the next chapter we will introduce an abstract machine model for deductive planning. It is in fact based on the machine model underlying the CHAM, which we have modified slightly for representing and handling planning problems.

Chapter 4

An Abstract Machine Model for Deductive Planning

4.1 Preliminary

In this chapter we will introduce a second example scenario and show how it can be solved within the extended equational logic programming framework. We will thereby once again demonstrate that the view of actions as resources is important if we want to model hierarchical planning. In particular, however, we want to give an answer to the question of how a deductive hierarchical planning system can be implemented.

One could, of course, run the programs on a standard PROLOG–system if the AC1–theory is translated into a suitable predicate and this predicate is called whenever two expressions are to be AC1–unified. Unfortunately, due to the inherent features of AC1–unification, this is far from being efficient. The obvious other choice is to map the equational logic programs onto a suitable abstract machine. Abstract machines, like the Turing Machine, are widely used in the classical theory of sequential and parallel computations. But what kind of abstract machine should we target for? The Warren Abstract Machine (WAM) [War83] and its variants are candidates. But, again, the usual search process carried out by a WAM has to be interleaved with the AC1–unification computation, and this causes considerable difficulties.

We therefore opted for an abstract machine, which has some sort of AC1–unification already built in, viz. the Chemical Abstract Machine (or CHAM, for short) [BB90]. Originally, the CHAM is a machine model for parallel processes. Molecules are floating within a solution, which is stirred by some mechanism. Some of the molecules may interact according to certain interaction rules. More formally, the solution is a multiset of molecules and the interaction rules are multiset rewritings (see Section 2.5).

The main result of this chapter is that the extended equational logic programming approach for reasoning about situations, actions, and causality can be mapped onto an extended version of the CHAM. This opens the door not only for efficiently implementing the equational logic programs, but also for applying actions concurrently. As mentioned, the CHAM is an abstract machine for parallel processes in that molecules may interact concurrently as long as the various interactions do not interfere.

4.2 Solving a Second Example

Traditionally, reasoning about situations, actions, and causality was modeled within a conceptual framework, where a situation is a snapshot of the world at a particular moment, and actions are the only means to transform one situation into another [McC63, MH69]. In the previous chapter, however, this framework was replaced by a more general framework based on a chemical metaphor first presented in [BB90]. There, a situation is represented as a *chemical solution* in which floating *molecules* can interact according to *interaction rules*. The molecules are terms representing either facts or actions, situations are finite multisets of molecules, and the interaction rules are multiset rewritings. Among the interaction rules are those, where actions are applied if their conditions are satisfied. But there are also more complex interactions like those, where two actions are combined to yield a more general action. To deal with such combinations as well as with abstraction and hierarchical planning, a molecule is allowed to contain a subsituation enclosed in a membrane, which can be somewhat porous to allow communication between the encapsulated situation and its environment.

To get another view on the extended ELP–approach we want to introduce a second example adapted from [Sac73], which is commonly known as the "standard" hierarchical planning example. Suppose an agent named Gisela comes home from work. All she wants is to relax and watch TV. The problem of watching TV can be described as a planning problem with the initial situation of Gisela being at the door (d) of the living room, the TV being unplugged (u), the actions *enter* and *turn–on–TV*, which enable Gisela to enter the living room (l) and to turn the TV on (on) respectively, and the goal situation where the TV is on. As we want to do hierarchical planning the action *turn–on–TV* can be refined to the actions *go–to–TV* for moving next to (n) the TV, *plug–in* for plugging the TV in (in), and *switch–on* for switching the TV on. One should observe that a further refinement of these operators is possible, but for the purpose of this work the above is in sufficient detail. An obvious solution to the problem on the highest level of abstraction is the plan containing the actions *enter* and *turn–on–TV*. A refinement of this plan contains the consecutive actions *go–to–TV*, *plug–in* and *switch–on* instead of the action *turn–on–TV*.

The formal model of the new approach towards modeling hierarchical planning in a deductive way has been summarized in Section 3.2.3. Furthermore, we have established that the equational logic program introduced in Section 3.2.4 handles hierarchical planning problems. The program requires SLDENF–resolution where the equational theory AC1 is built into the unification procedure and negative literals are solved via negation as failure [HT95a].[1] With the clauses in (3.4) and (3.19) for the predicates *transform* and *interact*, the equational theory AC1, and SLDENF–resolution we can now solve the planning problem presented above. Its initial situation described on the highest level of abstraction contains a primitive agent $\langle\emptyset.d\rangle$ representing the fact that Gisela is at the door of the living room, and two generators $!\langle d.l\rangle$, and $!\langle l.on \circ l\rangle$ for the actions *enter* and *turn–on–TV*. The whole problem is phrased as the following goal.

$$\leftarrow \mathit{transform}(\langle\emptyset.d\rangle \circ !\langle d.l\rangle \circ !\langle l.on \circ l\rangle, \langle\emptyset.on \circ Z\rangle \circ S). \qquad (4.1)$$

The variables Z and S occurring in the goal specification represent the other effects resulting from executing the generated plan and other resources still available, respectively. By applying the second

[1] One should observe that the only need for negation as failure is the test whether $D \neq_{AC1} \emptyset$ in the first clause of (3.19).

clause of (3.19) repeatedly to the two generators occurring in (4.1) using (3.4) we obtain

$$\leftarrow \mathit{transform}(\langle\emptyset.d\rangle \circ \langle d.l\rangle \circ \langle l.on \circ l\rangle \circ !\langle d.l\rangle \circ !\langle l.on \circ l\rangle, \langle\emptyset.on \circ Z\rangle \circ S).$$

The actions *enter* ($\langle d.l\rangle$) and *turn–on–TV* ($\langle l.on \circ l\rangle$) can now be successively applied to the fact $\langle\emptyset.d\rangle$ using the first clause of (3.19) and yielding

$$\leftarrow \mathit{transform}(\langle\emptyset.on \circ l\rangle \circ !\langle d.l\rangle \circ !\langle l.on\rangle, \langle\emptyset.on \circ Z\rangle \circ S).$$

From this goal the empty clause can be derived by applying the first clause of (3.4) obtaining the answer substitution $\{Z \mapsto l,\ S \mapsto !\langle d.l\rangle \circ !\langle l.on \circ l\rangle\}$. The generated plan itself can be extracted from the SLDENF–refutation and consists of the action *enter* followed by *turn–on–TV*.

So far we have solved the problem as if the actions *enter* and *turn–on–TV* were always available and could be executed by our agent Gisela. However, Gisela may be a robot and the actions that Gisela can perform are simpler than *turn–on–TV*. Hence, such a complex action must be synthesized from those simpler ones. Now assume that the simpler actions are the ones introduced in Section 4.2, i.e. the actions *go–to–TV* ($\langle l.n \circ l\rangle$), *plug–in* ($\langle u \circ n.in \circ n\rangle$), and *switch–on* ($\langle in \circ n.on \circ in \circ n\rangle$). It has to be shown, of course, that these simpler actions can indeed be combined to achieve the complex action. To do so, we replace the generator $!\langle l.on\rangle$ occurring in (4.1) by the membrane

$$M = \boxed{\langle\emptyset.u\rangle \circ !\langle l.n \circ l\rangle \circ !\langle u \circ n.in \circ n\rangle \circ !\langle in \circ n.on \circ in \circ n\rangle}\ .$$

In other words, we now assume that there is an unlimited supply of the simpler actions. We have also given some additional fact about the initial situation, viz. that the TV is unplugged ($\langle\emptyset.u\rangle$). Thus, we obtain the new goal

$$\leftarrow \mathit{transform}(\langle\emptyset.d\rangle \circ !\langle d.l\rangle \circ M, \langle\emptyset.on \circ Z\rangle \circ S). \tag{4.2}$$

We could solve this goal, if the complex action *turn–on–TV* could be generated by applying interaction rules within the membrane M. Once generated, the complex action can be pushed through the membrane with the help of the fifth and the sixth clause of (3.19) and, finally, the refutation can proceed as in the case of the refutation of (4.1). It is easy to see that applications of the second clause of (3.19) followed by applications of the first clause of (3.19) transforms the membrane M into

$$\boxed{\langle l.on \circ in \circ l \circ n\rangle \circ !\langle l.n \circ l\rangle \circ !\langle u \circ n.in \circ n\rangle \circ !\langle in \circ n.on \circ in \circ n\rangle}\ .$$

One should observe that the generated complex action $\langle l.on \circ in \circ l \circ n\rangle$ has additional effects compared to the action $\langle l.on \circ l\rangle$ used in the refutation of (4.1). This is no surprise as we have developed a more refined plan and in doing so more information about the conditions and effects of the involved actions becomes available.

One does not need to stop at this level. The action *go–to–TV*, for example, may itself be synthesized from even simpler actions which may require battery fuel for moving from one place of the living room to another one, etc. In this case, the generator $!\langle l.n \circ l\rangle$ has to be replaced by an appropriate membrane. In general, only the actions on the lowest level of abstraction are modeled as generators, whereas the actions on the other levels of abstraction are generated within membranes and then pushed through the membrane to the next higher level. In this way hierarchical planning can be achieved purely deductively.

4.3 The PCHAM — A CHAM for Deductive Planning

In this section we are going to transfer the functionality of the equational logic program introduced in the previous chapter (see (3.4) and (3.19)) to the machine model of the CHAM the basic principles of which we have introduced in Section 2.5. Before doing so we need to modify the CHAM slightly. Thereafter, we will show that the modified machine, which we will call PCHAM can handle planning problems as specified in Section 3.2.3, in particular we want to demonstrate how the example introduced in Section 4.2 can be solved. In other words, we show that the PCHAM is an appropriate abstract machine model for reasoning about situations, actions, and causality.

4.3.1 The Machine Model of the PCHAM

Let $\mathcal{N}$ be the set of terms denoting the facts as well as the conditions and effects of actions defined for a particular planning problem, and let α range over the set $\mathcal{V} = \{v^+, v^- : v \in \mathcal{N}\}$. Adding the symbol '!' (exponential) to the set of operators defined for a CHAM (see page 22), PCHAM molecules have the following form, where S denotes a multiset of molecules.

$$\begin{array}{rcl} m & ::= & \{p\} \mid !\{p\} \mid \{p\}\backslash^+ \mid \{p\}\backslash^- \mid \alpha.0 \lhd \{p\}\backslash^+ \mid \alpha.0 \lhd \{p\}\backslash^- \mid S \mid \{p\} \lhd S \\ \text{where } p & ::= & \alpha.0 \mid p|p \mid 0 \end{array}$$

We can adopt the parallel rule directly from Section 2.5; the reaction rule used by the PCHAM is slightly different from the reaction rule of the CHAM in that only ions[2] attached to an restricted airlock may perform reactions and, furthermore, two such ions may react if they are unifiable. The positive restriction operator ($\backslash^+$) ensures, that no positive valence (v^+) can be extracted from the encapsulated solution, likewise the negative restriction operator ($\backslash^-$) does not allow negative valences (v^-) to be extracted from the encapsulated solution.

$$\{v^- \lhd \{p_1\}\backslash^+, v^+ \lhd \{p_2\}\backslash^-\} \,\dot{\cup}\, S \;\rightarrow\; \sigma(\{\{p_1\}\backslash^+, \{p_2\}\backslash^-\} \,\dot{\cup}\, S) \qquad \text{(P–reaction)}$$

where σ is the most general unifier (mgu) of v^- and v^+. One should observe that the mgu of two terms itself can be computed by multiset rewritings (cf. [MM82]).

We slightly changed the airlock mechanism of the CHAM to fit our purposes. In the PCHAM it applies only to unrestricted membranes.

$$\begin{array}{rcll} \{\{p\}, m_1, m_2, \ldots, m_k\} & \rightleftharpoons & \{p\} \lhd \{m_1, m_2, \ldots, m_k\} & \text{(P–airlock)} \\ \{p\} \lhd \{m_1, m_2, \ldots, m_k\} & \rightleftharpoons & \{p\}, \{m_1, m_2, \ldots, m_k\} & \text{(disconnect)} \end{array}$$

Instead of the heavy ion rule there is a special airlock mechanism necessary, which extracts ions from a solution encapsulated in a restricted membrane.

$$\begin{array}{rcll} \{v^-, \alpha_1, \alpha_2, \ldots, \alpha_l\}\backslash^+ & \rightleftharpoons & v^- \lhd \{\alpha_1, \alpha_2, \ldots, \alpha_l\}\backslash^+ & \text{(P–ion)} \\ \{v^+, \alpha_1, \alpha_2, \ldots, \alpha_l\}\backslash^- & \rightleftharpoons & v^+ \lhd \{\alpha_1, \alpha_2, \ldots, \alpha_l\}\backslash^- & \end{array}$$

[2] Because a PCHAM ion $\alpha.0$ always has '0' in its body, we will denote it by α for notational convenience.

We need two rules for encapsulating solutions into restricted membranes and for removing restricted membranes again. A restricted membrane is a prerequisite for a reaction. Only pairs of molecules can be encapsulated in two complementary restricted membranes. Furthermore, to force a reaction to occur locally the two molecules become encapsulated in an additional membrane.

$$\begin{array}{rcll} \{\!\!\{p_1\}\!\!\}, \{\!\!\{p_2\}\!\!\} & \rightharpoonup & \{\!\!\{\{\!\!\{p_1\}\!\!\}\backslash^+, \{\!\!\{p_2\}\!\!\}\backslash^-\}\!\!\} & \text{(add restrict)} \\ \{\!\!\{\{\!\!\{p_1\}\!\!\}\backslash^+, \{\!\!\{p_2\}\!\!\}\backslash^-\}\!\!\} & \rightharpoondown & \{\!\!\{p_1, p_2\}\!\!\} & \text{(remove restrict)} \end{array}$$

Finally, we need an additional rule which handles the exponential $!$.

$$!\{\!\!\{p\}\!\!\} \;\rightarrow\; \{\!\!\{p\}\!\!\}, !\{\!\!\{p\}\!\!\} \qquad \text{(generator)}$$

4.3.2 Mapping ELP onto the PCHAM

In this subsection we want to show how SLDENF–refutations can be transformed into PCHAM–rewritings. Let i and g denote AC1-terms representing an initial and a goal situation, respectively. In particular, we will show that if there is an SLDENF–refutation from the goal $\leftarrow \mathit{transform}(i, g)$ wrt. the program consisting of the clauses in (3.4) and (3.19) with answer substitution θ, then there is a sequence of PCHAM rewriting steps from $\tau(i)$ to some g' such that $g' \doteq \tau(g\theta)$ holds, where τ transforms ELP–situations into PCHAM–situations.

Let X denote a variable, S denote a situation, $\mathcal{C}$ and $\mathcal{E}$ denote the conditions and effects of an agent, respectively, A denote an agent, and M_i denote molecules of any kind. The mapping τ from terms used to represent situations in ELP to the corresponding terms in the PCHAM is defined as follows:

$$\begin{array}{rcl@{\qquad}rcl} \tau(\emptyset) & = & \{\!\!\{\}\!\!\} & \tau^-(\emptyset) & = & 0 \\ \tau(X) & = & X & \tau^-(v) & = & v^- \\ \tau(\langle \mathcal{C}.\mathcal{E}\rangle) & = & \{\!\!\{\tau^-(\mathcal{C})|\tau^+(\mathcal{E})\}\!\!\} & \tau^-(v_1 \circ \ldots \circ v_n) & = & \tau^-(v_1)|\ldots|\tau^-(v_n) \\ \tau(!\langle \mathcal{C}.\mathcal{E}\rangle) & = & !\ \tau(\langle \mathcal{C}.\mathcal{E}\rangle) & \tau^+(\emptyset) & = & 0 \\ \tau(\boxed{S}) & = & \tau(S) & \tau^+(v) & = & v^+ \\ \tau(A \lhd \boxed{S}) & = & \tau(A) \lhd \tau(\boxed{S}) & \tau^+(v_1 \circ \ldots \circ v_n) & = & \tau^+(v_1)|\ldots|\tau^+(v_n) \end{array}$$

$$\begin{array}{rcl} \tau(M_1 \circ \ldots \circ M_n) & = & \{\!\!\{\tau(M_1), \ldots, \tau(M_n)\}\!\!\} \\ \tau(M_i \circ \ldots \circ M_n \circ X) & = & \{\!\!\{\tau(M_1 \circ \ldots \circ M_n)\}\!\!\} \mathbin{\dot\cup} \tau(X) \end{array}$$

Having specified molecules and solutions of the PCHAM we can now turn our attention to the mapping of SLDENF–derivations onto PCHAM–rewritings. The goal is to give a sequence of PCHAM–rewriting steps for each application of SLDENF–resolution using one of the clauses of (3.19). As we will demonstrate, the PCHAM–rewriting steps will consider only that part of a situation which changes, but not the context. Hence, PCHAM–rewritings solve the frame problem in a quite natural way.

The first clause of (3.19) defines how a reaction between two agents $\langle \mathcal{C}_1.\mathcal{E}_1\rangle$ and $\langle \mathcal{C}_2.\mathcal{E}_2\rangle$ in a solution takes place. Let $\{\!\!\{c^-_{1,1}|\ldots|c^-_{1,k}|e^+_{1,1}|\ldots|e^+_{1,l}\}\!\!\}$ be $\tau(\langle \mathcal{C}_1.\mathcal{E}_1\rangle)$, and $\{\!\!\{c^-_{2,1}|\ldots|c^-_{2,m}|e^+_{2,1}|\ldots|e^+_{2,n}\}\!\!\}$ be $\tau(\langle \mathcal{C}_2.\mathcal{E}_2\rangle)$, respectively. The corresponding sequence of PCHAM-transformation rules[3] is stated below,

[3] By $\xrightarrow{x}$ we denote that the interaction rule was applied x–times.

where we have abbreviated $c^-_{1,1},\ldots,c^-_{1,k}$ by $\overline{c}$ and $e^+_{2,1},\ldots,e^+_{2,n}$ by $\overline{e}$.

$$
\begin{array}{clr}
 & \{c^-_{1,1}|\ldots|c^-_{1,k}|e^+_{1,1}|\ldots|e^+_{1,l}\},\{c^-_{2,1}|\ldots|c^-_{2,m}|e^+_{2,1}|\ldots|e^+_{2,n}\} & \\
\overset{*}{\rightharpoonup} & \{\overline{c},e^+_{1,1},\ldots,e^+_{1,l}\},\{c^-_{2,1},\ldots,c^-_{2,m},\overline{e}\} & \text{(parallel)}\\
\rightharpoonup & \{\{\overline{c},e^+_{1,1},\ldots,e^+_{1,l}\}\backslash^-,\{c^-_{2,1},\ldots,c^-_{2,m},\overline{e}\}\backslash^+\} & \text{(add restrict)}\\
\overset{2}{\rightharpoonup} & \{e^+_{1,1}\lhd\{\overline{c},e^+_{1,2},\ldots,e^+_{1,l}\}\backslash^-,c^-_{2,1}\lhd\{c^-_{2,2},\ldots,c^-_{2,m},\overline{e}\}\backslash^+\} & \text{(P–ion)}\\
\to & \sigma_1(\{\{\overline{c},e^+_{1,2},\ldots,e^+_{1,l}\}\backslash^-,\{c^-_{2,2},\ldots,c^-_{2,m},\overline{e}\}\backslash^+\}) & \text{(P–reaction)}\\
\overset{*}{\to} & \sigma_d\ldots\sigma_1(\{\{\overline{c},e^+_{1,d+1},\ldots,e^+_{1,l}\}\backslash^-,\{c^-_{2,d+1},\ldots,c^-_{2,m},\overline{e}\}\backslash^+\}) & \\
\rightharpoondown & \sigma_d\ldots\sigma_1(\{\overline{c},e^+_{1,d+1},\ldots,e^+_{1,l},c^-_{2,d+1},\ldots,c^-_{2,m},\overline{e}\}) & \text{(remove restrict)}\\
\overset{*}{\rightharpoondown} & \sigma_d\ldots\sigma_1(\{c^-_{1,1}|\ldots|c^-_{1,k}|e^+_{1,d+1}|\ldots|e^+_{1,l}|c^-_{2,d+1}|\ldots|c^-_{2,m}|e^+_{2,1}|\ldots|e^+_{2,n}\}) & \text{(parallel)}
\end{array}
$$

One should observe that in the step (P–ion) any molecule with a positive or negative valence can be pushed through the membrane with negative or positive restriction, respectively.

In the sequel we want to abbreviate the above sequence of rules by the following macro–rule.

$$
\begin{array}{l}
\{c^-_{1,1}|\ldots|c^-_{1,k}|e^+_{1,1}|\ldots|e^+_{1,l}\},\ \{c^-_{2,1}|\ldots|c^-_{2,m}|e^+_{2,1}|\ldots|e^+_{2,n}\}\overset{M}{\to}\\
\qquad\qquad \sigma(\{c^-_{1,1}|\ldots|c^-_{1,k}|e^+_{1,d+1}|\ldots|e^+_{1,l}|c^-_{2,d+1}|\ldots|c^-_{2,m}|e^+_{2,1}|\ldots|e^+_{2,n}\}) \qquad \text{(macro)}\\
\text{with } \exists v^+,v^-.\ [v^+=e^+_{1,i}\ \wedge\ v^-=c^-_{2,j}\ ] \text{ and } 1\le i\le d, 1\le j\le d, \text{ where } 1\le d\le \min(l,m)
\end{array}
$$

where d denotes the number of valences vanishing in the reaction, and σ is the iteratively produced most general unifier during the performance of these reactions. It should be noted that the only need of the negation as failure part within an SLDENF–refutation corresponds to the fact that within the macro sequence at least one P–reaction has to take place.

The second clause of (3.19) is used to generate new agents. The corresponding interaction rule, viz. the generator rule mimics this clause. As the chemical law (L2) and the membrane law (L3) require that interactions can be performed freely within any solution and context, these laws take care of the third and forth clause of (3.19). Finally, the remaining clauses of (3.19), which are the fifth and sixth clause, are handled by the P–airlock rule and the disconnection rule of the PCHAM.

Within the next paragraphs we want to deeper investigate the connection between an SLDENF–refutation of a goal wrt. the program consisting of the clauses in (3.4) and (3.19), and its corresponding sequence of PCHAM states. It is obvious that only derivation steps using one of the clauses belonging to the definition of the predicate *interact*, result in a non-empty sequence of PCHAM machine states. We therefore state the following lemma:

Lemma 4 *Let s_1 and s_2 denote AC1–terms representing situations, and let s_1 be ground. If there is an SLDENF–refutation of $\{\leftarrow interact(s_1,s_2)\}\cup\{(3.19)\}$ with computed answer substitution σ, then there is a PCHAM–rewriting from $\tau(s_1)$ into some situation $\hat{s_2}$ such that $\hat{s_2}\doteq\tau(s_2\sigma)$ holds.*

Proof
The proof is by transfinite induction on the length l of the SLDENF–refutation. Suppose the result holds for all SLDENF–refutations of length less than l. Let

$$\leftarrow interact(s_1, s_2) \xrightarrow{l}_{\text{SLDENF}} \Box$$

be an SLDENF–refutation of length l. Furthermore, let P be the program clause and λ be the unifier used in the first step of this refutation. Finally, let G be the resolvent of the initial goal and P using λ, i.e.,

$$\leftarrow interact(s_1, s_2) \xrightarrow{1}_{\text{SLDENF}} G \xrightarrow{l-1}_{\text{SLDENF}} \Box$$

We distinguish six cases wrt. P.

1. If $P \equiv interact(!\langle C.E\rangle \circ Z, \langle C.E\rangle \circ !\langle C.E\rangle \circ Z)$ then G is the empty clause and $l = 1$. Furthermore, we find

$$!\langle C\lambda.E\lambda\rangle \circ Z\lambda \ =_{AC1}\ s_1\lambda = s_1$$

and

$$\langle C\lambda.E\lambda\rangle \circ !\langle C\lambda.E\lambda\rangle \circ Z\lambda \ =_{AC1}\ s_2\lambda = s_2\sigma.$$

Hence,

$$\tau(s_1) = \{\!|\, !\tau(\langle C\lambda.E\lambda\rangle) \,|\!\} \ \dot{\cup}\ \tau(Z\lambda) \tag{4.3}$$

and

$$\tau(s_2\sigma) = \tau(s_2\lambda) = \{\!|\, !\tau(\langle C\lambda.E\lambda\rangle), \tau(\langle C\lambda.E\lambda\rangle) \,|\!\} \ \dot{\cup}\ \tau(Z\lambda). \tag{4.4}$$

Now, obviously, (4.3) $\rightarrow$ (4.4) with the *generator* rule, which concludes this case.

2. If $P \equiv interact(Z \circ \boxed{\langle C.E\rangle \circ S}, Z \circ \langle C.E\rangle \lhd \boxed{S})$ then G is the empty clause and $l = 1$. Furthermore, we find

$$Z\lambda \circ \boxed{\langle C\lambda.E\lambda\rangle \circ S\lambda} \ =_{AC1}\ s_1\lambda = s_1$$

and

$$Z\lambda \circ \langle C\lambda.E\lambda\rangle \lhd \boxed{S\lambda} \ =_{AC1}\ s_2\lambda = s_2\sigma.$$

Hence,

$$\tau(s_1) = \tau(Z\lambda) \ \dot{\cup}\ \{\ \tau(\langle C\lambda.E\lambda\rangle) \ \dot{\cup}\ \tau(S\lambda)\ \} \tag{4.5}$$

and

$$\tau(s_2\sigma) = \tau(s_2\lambda) = \tau(Z\lambda) \ \dot{\cup}\ \{\ \tau(\langle C\lambda.E\lambda\rangle) \lhd \tau(S\lambda)\ \}. \tag{4.6}$$

Now, obviously, (4.5) $\rightharpoonup$ (4.6) with the *P-airlock* rule, which concludes this case.

3. If $P \equiv interact(Z \circ \langle C.E\rangle \lhd \boxed{S}, Z \circ \langle C.E\rangle \circ \boxed{S})$ then G is the empty clause and $l = 1$. Furthermore, we find

$$Z\lambda \circ \langle C\lambda.E\lambda\rangle \lhd \boxed{S\lambda} \ =_{AC1}\ s_1\lambda = s_1$$

and

$$Z\lambda \circ \langle C\lambda.E\lambda\rangle \circ \boxed{S\lambda} \ =_{AC1}\ s_2\lambda = \sigma s_2.$$

Hence,

$$\tau(s_1) = \tau(Z\lambda) \ \dot{\cup}\ \{\ \tau(\langle C\lambda.E\lambda\rangle) \lhd \tau(S\lambda)\ \} \tag{4.7}$$

and

$$\tau(s_2\sigma) = \tau(s_2\lambda) = \tau(Z\lambda) \mathbin{\dot\cup} \{\!\!\{ \ \tau(\langle C\lambda.E\lambda\rangle), \tau(S\lambda) \ \}\!\!\}. \tag{4.8}$$

Now, obviously, (4.7) $\rightharpoonup$ (4.8) with the *disconnect* rule, which concludes this case.

4. If $P \equiv interact(\langle C_1.E_1 \circ D\rangle \circ \langle C_2 \circ D.E_2\rangle \circ Z,\ \langle C_1 \circ C_2.E_1 \circ E_2\rangle \circ Z) \leftarrow D \neq_{AC1} \emptyset$ then $G \equiv \leftarrow D\lambda \neq_{AC1} \emptyset$ and $l = 2$. Furthermore, we find

$$\langle C_1\lambda.E_1\lambda \circ D\lambda\rangle \circ \langle C_2\lambda \circ D\lambda.E_2\lambda\rangle \circ Z\lambda \ =_{AC1}\ s_1\lambda = s_1$$

and

$$\langle C_1\lambda \circ C_2\lambda.E_1\lambda \circ E_2\lambda\rangle \circ Z\lambda \ =_{AC1}\ s_2\lambda = s_2\sigma.$$

Hence,

$$\tau(s_1) = \{\!\!\{\tau(\langle C_1\lambda.E_1\lambda \mathbin{\dot\cup} D\lambda\rangle), \tau(\langle C_2\lambda \mathbin{\dot\cup} D\lambda.E_2\lambda\rangle)\}\!\!\} \mathbin{\dot\cup} \tau(Z\lambda) \tag{4.9}$$

and

$$\tau(s_2\sigma) = \tau(s_2\lambda) = \{\!\!\{\tau(\langle C_1\lambda \mathbin{\dot\cup} C_2\lambda.E_1\lambda \mathbin{\dot\cup} E_2\lambda\rangle)\}\!\!\} \mathbin{\dot\cup} \tau(Z\lambda). \tag{4.10}$$

Now, obviously (4.9) $\stackrel{M}{\rightarrow}$ (4.10) with the *macro* rule. Recall that the *macro* rule requires at least one valence vanishing during the reaction, thus a check whether $D\lambda \neq_{AC1} \emptyset$ is implicitly contained in the application of the *macro* rule, which concludes this case.

5. If $P \equiv interact(Z \circ \boxed{S_1'}, Z \circ \boxed{S_2'}) \leftarrow interact(S_1', S_2')$ then $G \ \equiv \leftarrow interact(S_1'\lambda, S_2'\lambda)$. Let σ' be the computed answer substitution of the refutation of $\{G\} \cup \{(3.19)\}$, i.e. $\sigma = \lambda\sigma'|_{\mathcal{V}ar(s_2)}$. Furthermore, we find

$$Z\lambda \circ \boxed{S_1'\lambda} \ =_{AC1}\ s_1\lambda = s_1 \tag{4.11}$$

and

$$Z\lambda \circ \boxed{S_2'\lambda} \ =_{AC1}\ s_2\lambda. \tag{4.12}$$

Hence,

$$\tau(s_1) = \tau(Z\lambda) \mathbin{\dot\cup} \{\!\!\{\tau(S_1'\lambda)\}\!\!\} \tag{4.13}$$

and

$$\tau(s_2\lambda) = \tau(Z\lambda) \mathbin{\dot\cup} \{\!\!\{\tau(S_2'\lambda)\}\!\!\}. \tag{4.14}$$

From (4.11) and the fact that the equational theory AC1 is regular we conclude that $S_1'\lambda$ is ground. Hence, by an application of the induction hypotheses to the refutation of G we find a situation $\hat{s_2'}$ and a PCHAM rewriting

$$\tau(S_1'\lambda) \stackrel{*}{\rightarrow} \hat{s_2'} \tag{4.15}$$

such that $\hat{s_2'} \doteq \tau(S_2'\lambda\sigma')$ holds. As $\{\!\!\{ \ \tau(S_1'\lambda) \ \}\!\!\}$ is a submultiset of $\tau(s_1)$ (by (4.13)) and the PCHAM obeys the chemical (L2) and the membrane law (L3), we conclude from (4.15) that there is a PCHAM rewriting

$$\tau(s_1) \stackrel{*}{\rightarrow} \tau(Z\lambda\sigma') \mathbin{\dot\cup} \{\!\!\{\hat{s_2'}\}\!\!\}$$

The result follows immediately from the fact that $\hat{s_2'} \doteq \tau(S_2'\lambda\sigma')$ holds, from $\sigma = \lambda\sigma'|_{\mathcal{V}ar(s_2)}$ and (4.12), (4.14), which concludes this case.

6. If $P \equiv interact(Z \circ \langle C.E\rangle \lhd \boxed{S_1'}, Z \circ \langle C.E\rangle \lhd \boxed{S_2'}) \leftarrow interact(S_1', S_2')$ then $G \equiv \leftarrow interact(S_1'\lambda, S_2'\lambda)$. Let σ' be the computed answer substitution of the refutation of $\{G\} \cup \{(3.19)\}$, i.e. $\sigma = \lambda\sigma'|_{\mathcal{V}ar(s_2)}$. Furthermore, we find

$$Z\lambda \circ \langle C\lambda.E\lambda\rangle \lhd \boxed{S_1'\lambda} =_{AC1} s_1\lambda = s_1 \tag{4.16}$$

and

$$Z\lambda \circ \langle C\lambda.E\lambda\rangle \lhd \boxed{S_2'\lambda} =_{AC1} s_2\lambda. \tag{4.17}$$

Hence,

$$\tau(s_1) = \tau(Z\lambda) \mathbin{\dot\cup} \{\!\!\{\tau(\langle C\lambda.E\lambda\rangle) \lhd \tau(S_1'\lambda)\}\!\!\} \tag{4.18}$$

and

$$\tau(s_2\lambda) = \tau(Z\lambda) \mathbin{\dot\cup} \{\!\!\{\tau(\langle C\lambda.E\lambda\rangle) \lhd \tau(S_2'\lambda)\}\!\!\}. \tag{4.19}$$

From (4.16) and the fact that the equational theory AC1 is regular we conclude that $S_1'\lambda$ is ground. Hence, by an application of the induction hypotheses to the refutation of G we find a situation $\hat{s}_2'$ and a PCHAM rewriting

$$\tau(S_1'\lambda) \xrightarrow{*} \hat{s}_2' \tag{4.20}$$

such that $\hat{s}_2' \doteq \tau(S_2'\lambda\sigma')$ holds. As $\tau(S_1'\lambda)$ is in the context of an airlock contained in a submultiset of $\tau(s_1)$ (by (4.18)) and the PCHAM obeys the chemical (L2) and the membrane law (L3), we conclude from (4.20) that there is a PCHAM rewriting

$$\tau(s_1) \xrightarrow{*} \tau(Z\lambda\sigma') \mathbin{\dot\cup} \{\!\!\{\tau(\langle C\lambda\sigma'.E\lambda\sigma'\rangle) \lhd \hat{s}_2'\}\!\!\}$$

The result follows immediately from the fact that $\hat{s}_2' \doteq \tau(S_2'\lambda\sigma')$ holds, from $\sigma = \lambda\sigma'|_{\mathcal{V}ar(s_2)}$ and (4.17), (4.19), which concludes this case. ■

The proof shows that by performing the transformations we obey the chemical law (L2) and the membrane law (L3). The term represented by the variable Z is not affected by the application of any of the clauses in (3.19). Note that this is how we solve the frame problem in the equational logic programming approach, whereas solving the frame problem in the PCHAM is based on the fact that the interaction rules can be applied anywhere in a solution according to the chemical law (L2) and the membrane law (L3).

The following theorem establishes that the PCHAM can handle the planning problems specified in Section 3.2.3.

Theorem 5 *Let i and g denote AC1–terms representing situations, and let i be ground. If there is an SLDENF–refutation of $\{\leftarrow \mathrm{transform}(i, g)\} \cup \{(3.4), (3.19)\}$ with the computed answer substitution θ then there is PCHAM–rewriting from $\tau(i)$ to some situation $\hat{g}$, such that $\hat{g} \doteq \tau(g\theta)$ holds.*

Proof
The theorem is proved by transfinite induction on the length l of the SLDENF–refutation using the correspondence between SLDENF–resolution steps and PCHAM–rewriting steps mentioned before.

Suppose the theorem holds for all SLDENF-refutations of length less than l. Let

$$\mathit{transform}(i, g) \xrightarrow{l}_{\text{SLDENF}} \Box$$

be an SLDENF–refutation of length l. Furthermore, let P be the program clause and λ be the unifier used in the first step of the above refutation. Finally, let R be the resolvent of the initial goal and P using λ, i.e.

$$\mathit{transform}(i, g) \xrightarrow{1}_{\text{SLDENF}} R \xrightarrow{l-1}_{\text{SLDENF}} \Box$$

We distinguish the following two cases:

1. If $P \equiv \mathit{transform}(I, I)$ then R is the empty clause and $l = 1$. We find
$$I\lambda =_{AC1} i\lambda = i \tag{4.21}$$
and
$$I\lambda =_{AC1} g\lambda. \tag{4.22}$$
Hence,
$$\tau(i) = \tau(I\lambda) \tag{4.23}$$
and
$$\tau(g\lambda) = \tau(I\lambda). \tag{4.24}$$
From (4.21) and (4.22) it follows that $i =_{AC1} g\lambda$, and from (4.23) and (4.24) it follows that $\tau(i) \doteq \tau(g\lambda)$, which in fact means that $\hat{g} \doteq \tau(i) \doteq \tau(g\lambda)$. Consequently the corresponding PCHAM–rewriting is empty.

2. If $P \equiv \mathit{transform}(I, G) \leftarrow \mathit{interact}(I, S) \wedge \mathit{transform}(S, G)$ then $R \equiv \leftarrow \mathit{interact}(I\lambda, S\lambda) \wedge \mathit{transform}(S\lambda, G\lambda)$. Let σ be the computed answer substitution of the refutation of the first subgoal of R, and θ' be the computed answer substitution of the refutation of the last subgoal of R, i.e. $\theta = \lambda\sigma\theta'|_{\mathcal{V}ar(g)}$. Moreover, we find
$$I\lambda =_{AC1} i\lambda = i \tag{4.25}$$
and
$$G\lambda =_{AC1} g\lambda. \tag{4.26}$$
Hence,
$$\tau(i) = \tau(I\lambda) \tag{4.27}$$
and
$$\tau(g\lambda) = \tau(G\lambda). \tag{4.28}$$
From (4.25) and the fact that the equational theory AC1 is regular it follows that $I\lambda$ is ground. By application of Lemma 3 to the refutation of the first subgoal $\leftarrow \mathit{interact}(I\lambda, S\lambda)$ of R we find a situation $\hat{s}$ such that $\hat{s} \doteq \tau(S\lambda\sigma)$ holds, and the following PCHAM–rewriting
$$\tau(I\lambda) \xrightarrow{*}_{\text{PCHAM}} \hat{s}. \tag{4.29}$$

The equational theory AC1 is regular, and $I\lambda$ is ground. Furthermore, for each head $interact(s_1, s_2)$ of the clauses defined in (3.19) we find $\mathcal{V}ar(s_1) = \mathcal{V}ar(s_2)$. Since $I\lambda$ is ground, i.e. $\mathcal{V}ar(I\lambda) = \emptyset$, and we solved the goal $\leftarrow interact(I\lambda, S\lambda)$, we consequently conclude that $S\lambda\sigma$ is ground. Therefore we can apply the induction hypothesis to the refutation of the last subgoal $\leftarrow transform(S\lambda\sigma, G\lambda\sigma)$ of R and find a situation $\hat{g}$ and the PCHAM–rewriting

$$\tau(S\lambda\sigma) \xrightarrow{*}_{\text{PCHAM}} \hat{g}. \tag{4.30}$$

The result follows immediately from (4.25) and (4.27), the fact that $\hat{g} \doteq \tau(g\lambda\sigma\theta')$ holds, from $\theta = \lambda\sigma\theta'|_{\mathcal{V}ar(g)}$, (4.26) and (4.28). The concatenation of the two PCHAM–rewritings in (4.29) and (4.30) yields a PCHAM–rewriting from $\tau(i)$ to $\hat{g}$. ■

In the following we depict the PCHAM–rewriting sequences for the example discussed in Section 4.2. The first sequence corresponds to the SLDENF-derivation of the goal (4.1). We have underlined the interacting molecules.

$$\begin{aligned}
& \{ \{0|d^+\},\ \underline{!\{d^-|l^+\}},\ \underline{!\{l^-|on^+|l^+\}} \} \\
\overset{2}{\rightarrow}\ & \{ \underline{\{0|d^+\}},\ \underline{\{d^-|l^+\}},\ \{l^-|on^+|l^+\},\ !\{d^-|l^+\},\ !\{l^-|on^+|l^+\} \} && \text{(generator)} \\
\overset{M}{\rightarrow}\ & \{ \underline{\{0|l^+\}},\ \underline{\{l^-|on^+|l^+\}},\ !\{d^-|l^+\},\ !\{l^-|on^+|l^+\} \} \\
\overset{M}{\rightarrow}\ & \{ \{0|on^+|l^+\},\ !\{d^-|l^+\},\ !\{l^-|on^+|l^+\} \}
\end{aligned}$$

It is easy to see that the final state of the PCHAM corresponds to the instantiated goal state of (4.1) modulo AC1.

The second sequence corresponds to the SLDENF–refutation of (4.2) except that we have modeled the action *enter* as an agent and have applied it first. It shows how membranes can be used to model the lower levels of the planning hierarchy. Let T denote the unlimited source of actions at the lower level, i.e.

$$T = !\{l^-|n^+|l^+\},\ !\{u^-|n^-|in^+|n^+\},\ !\{in^-|n^-|on^+|in^+|n^+\}.$$

$$\begin{aligned}
& \{ \underline{\{0|d^+\}},\ \underline{\{d^-|l^+\}},\ \{ \{0|u^+\},\ T\} \} \\
\overset{M}{\rightarrow}\ & \{ \{0|l^+\},\ \{ \{0|u^+\},\ \underline{T}\} \} \\
\overset{3}{\rightarrow}\ & \{ \{0|l^+\},\ \{ \underline{\{0|u^+\}},\ \{l^-|n^+|l^+\},\ \underline{\{u^-|n^-|in^+|n^+\}},\ \{in^-|n^-|on^+|in^+|n^+\},\ T\} \} \\
& && \text{(generator)} \\
\overset{M}{\rightarrow}\ & \{ \{0|l^+\},\ \{ \underline{\{l^-|n^+|l^+\}},\ \underline{\{n^-|in^+|n^+\}},\ \{in^-|n^-|on^+|in^+|n^+\},\ T\} \} \\
\overset{M}{\rightarrow}\ & \{ \{0|l^+\},\ \{ \underline{\{l^-|in^+|l^+|n^+\}},\ \underline{\{in^-|n^-|on^+|in^+|n^+\}},\ T\} \} \\
\overset{M}{\rightarrow}\ & \{ \{0|l^+\},\ \{ \underline{\{l^-|on^+|in^+|n^+|l^+\}},\ T\} \} \\
\rightarrow\ & \{ \{0|l^+\},\ \underline{\{l^-|on^+|in^+|n^+|l^+\}} \lhd \{ T\} \} && \text{(P–airlock)} \\
\rightarrow\ & \{ \underline{\{0|l^+\}},\ \underline{\{l^-|on^+|in^+|n^+|l^+\}},\ \{ T\} \} && \text{(disconnect)} \\
\overset{M}{\rightarrow}\ & \{ \{ 0|on^+|in^+|n^+|l^+\},\ \{ T\} \}
\end{aligned}$$

With these two examples we have demonstrated the basic functionality of the PCHAM. We believe that the abstract machine model of the PCHAM can be further refined especially in view of a parallel computational model for reasoning about situations, actions, and causality. In the next chapter we will summarize our results and give an outlook on possible further research topics and refinements of the approach introduced in the previous two chapters.

Chapter 5

Conclusion and Outlook on Further Research

In this chapter we first want to summarize the results of this work and draw conclusions from the experiences whilst doing the research. Finally, since the results of this thesis are only a first step towards a new direction of resource–oriented modeling, we want to give some ideas of how to extend our new approach.

5.1 Conclusion

In the last couple of years we have seen several new approaches for reasoning about situations, actions, and causality, which solve the frame problem without the need to state frame axioms or similar laws of inertia. In contrast to conventional approaches like the situation calculus, where the facts about a situation are modeled as properties, these new approaches model the facts about a situation as resources.

The aim of this thesis was to develop an extension to ELP [HS90] which handles resource-oriented hierarchical planning in a purely deductive way. The approach is based on the observation that the frame problem can be solved within a purely declarative framework if facts about a situation are represented as resources which are consumed and produced by actions. The major new idea presented in this work is the view of actions as resources in a resource–oriented theory of actions. This is motivated by the fact that in certain scenarios — like the tea–scenario — the partial fulfillment of conditions of actions or the combination of actions could not be adequately modeled within the old ELP–framework. As shown, such scenarios and, more generally, abstractions and hierarchical planning can be adequately modeled if actions itself are viewed as resources which are produced and consumed just as facts about a situation can be consumed and produced.

By hierarchical planning we understand a method for reducing the search space in problem solving in a particular scenario by successive refinement of the description of the problem space. As a result the original problem space is stratified into different levels of abstraction. Solving the problem consists of finding a solution on the highest abstraction level; subsequently refining the problem space by introducing more detailed information on the next lower abstraction levels, solving subproblems, and integrating the solutions found into the overall solution of the problem until the lowest abstraction level is reached, which means that a solution to the whole problem has been found.

In the new concept facts about a situation as well as actions are represented as agents which interact by certain interaction rules. To model an unlimited source of actions we have introduced generators which in fact have a similar functionality to the concept of an exponential in linear logic [Gir87]. Concerning hierarchical planning the various levels within a hierarchy are modeled with the help of membranes, and the interactions between levels are achieved via the airlock mechanism. Both these concepts are adopted from the Chemical Abstract Machine [BB90]. Consequently, a situation consists of a collection of facts and available actions. It is structured into subsituations by means of membranes, and interaction rules transform one situation into another. As the interaction rules specify not only the application of actions, the model is of a somehow finer grain than the situation calculus.

The actual refinement needed to do hierarchical planning is achieved by replacing a generator for a complex action by a membrane containing simpler actions and generators, which again react to form a refined more complex action. This action is pushed through the membrane to the next higher level via the airlock mechanism and can react freely in its new environment.

With two examples we have demonstrated that the new approach is suitable for solving both conventional as well as hierarchical planning problems. Hence we have shown that the new approach is not only a conservative extension of the original ELP–approach, but it is also the basis for a purely declarative model for abstractions and hierarchical planning. Furthermore, we have shown that the solutions for hierarchical planning problems can be obtained deductively as SLDENF–refutations.

As the concept of a membrane was borrowed from the CHAM, the next question to answer was how the hierarchical planning approach was related to the CHAM. In answering this question it turned out that the abstract machine model underlying the CHAM is a quite natural concept for reasoning about situations, actions, and causality. Consequently, we developed a new abstract machine, called PCHAM, which extends the CHAM. The main feature of this extension is the incorporation of logical variables into the CHAM by means of the P–reaction rule. This is a simple extension of the CHAM as the most general unifier of two expressions can be computed by multiset rewritings. We showed that the PCHAM can serve as an abstract machine for the kind of reasoning problems considered in this work.

In the following we are going to give some ideas on how the approach proposed within this work can be extended in order to gain even better expressive power.

5.2 Outlook on Further Research

In the future the new approach should be further developed into various directions.

Firstly, the approach does not specify the order in which interaction rules are to be applied. As the interaction rules are fine–grained, many of them are usually applicable at each point of time. We do need to find adequate heuristics for the application of these rules.

Secondly, if the scenario is not as simple as the ones presented herein, then interaction rules may be applied in parallel at various levels of the specified hierarchy. Thus, we should be able to model multi–agent planning problems. Obviously, one major problem is to avoid conflicts which occur as soon as two agents are attempting to use a certain resource at the same time. Another major conflict is how these agents should work together in order to achieve a common goal.

Thirdly, so far we have dealt only with the frame problem. For resource–oriented approaches to theories of actions like ELP [HS90], the linear connection method [Bib86], linear logic [MTV90], or the approach presented in this work, the frame problem is in fact no problem. The resources which are not affected by an action just remain as they are when the action is applied. But we have to deal with the other problems, viz. the ramification, qualification, and prediction problem as well.

Fourthly, the plans developed within the model are linear plans and the approach should be extend to handle partially–ordered plans as well. In the future we should also investigate recurrent scenarios.

Furthermore, there are several technical problems too. E.g., so far we have not specified which agents can be pushed through a membrane. This could be achieved by turning the unary function representing a membrane into a binary one, and using the additional argument to specify templates which have to be matched by agents before being pushed through the membrane. Furthermore, recall that we have restricted actions to ground ones. We may lift this restriction without any problems but one. If a generator produces a copy of its agent, then we have to ensure that the variables occurring in the copy are standardized apart. As agents are terms, this problem has its counterpart in meta–programming, where the copies of program clauses need also to be standardized apart.

Also, we want to incorporate the various extensions to ELP discussed in Chapter 2 into the new approach. We believe that e.g. the concept of specificity (see Section 2.3.3) can easily be adapted. The basic idea is to provide more specific agents and less specific agents in the same membrane. As long as there are more specific agents available for performing reactions, the less specific ones are not considered. Informally an agent is more specific than another agent if it has more (detailed) conditions, i.e. the conditions of the latter are a proper subset of the conditions of the former.

Finally, the abstract machine model developed in this work opens a new line of research. In the rewriting sequences only two molecules were allowed to react at a given time. But there is absolutely no reason why not more than a single pair of molecules may interact at a given time as long as the various interactions do not interfere. For example, in the final rewriting sequence of the previous Chapter 4 the first four rewriting steps ($\stackrel{M}{\rightarrow}, \stackrel{3}{\rightarrow}$) could have been performed in parallel. Hence, we may employ the experience gathered with the CHAM to develop a multi–agent deductive planning system. In fact, the CHAM was devised as a machine model for asynchronous concurrent computations. We envision that in a larger world model many local interactions may take place at a time and that the overall global goal is achieved by means of these local interactions. Again, one of the major problems to be solved for such concurrent computational models is the avoidance of conflicts and to ensure that independent agents work together in order to achieve a global goal. Both problems are essentially instances of the question in which order the interaction rules should be applied, which we have mentioned earlier on.

Bibliography

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

[Bar81] H. Barendregt. The type-free Lambda-Calculus. Studies in Logic Volume 103, North Holland, 1981.

[BB90] G. Berry and G. Boudol. The Chemical Abstract Machine. In *Proceedings of the ACM Symposium on Principles of Programming Languages*, pages 81–94, 1990.

[BG93] C. Baral and M. Gelfond. Representing Concurrent Actions in Extended Logic Programming. In R. Bajcsy, editor, *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 866–871, Chambéry, France, August 1993. Morgan Kaufmann.

[BHS+93] S. Brüning, S. Hölldobler, J. Schneeberger, U. Sigmund, and M. Thielscher. Disjunction in Resource-Oriented Deductive Planning. In D. Miller, editor, *Proceedings of the International Logic Programming Symposium (ILPS)*, page 670, Vancouver, Canada, October 1993. MIT Press. (Poster).

[Bib86] W. Bibel. A deductive solution for plan generation. *New Generation Computing*, 4:115–132, 1986.

[BT94] S.-E. Bornscheuer and M. Thielscher. Representing Concurrent Actions and Solving Conflicts. In B. Nebel and L. Dreschler-Fischer, editors, *Proceedings of the German Annual Conference on Artificial Intelligence (KI)*, volume 861 of *LNAI*, pages 16–27, Saarbrücken, Germany, September 1994. Springer.

[Cla77] K. L. Clark. Negation as failure. In Gallaire and Nicolas, editors, *Workshop Logic and Databases*, CERT, Toulouse, France, 1977.

[Cla78] K. L. Clark. Negation as Failure. In H. Gallaire and J. Minker, editors, *Workshop Logic and Data Bases*, pages 293–322. Plenum Press, 1978.

[DdS93] M. Denecker and D. de Schreye. Representing Incomplete Knowledge in Abductive Logic Programming. In D. Miller, editor, *Proceedings of the International Logic Programming Symposium (ILPS)*, pages 147–163, Vancouver, October 1993. MIT Press.

[Dun93] P. M. Dung. Representing Actions in Logic Programming and its Applications in Database Updates. In D. S. Warren, editor, *Proceedings of the International Conference on Logic Programming (ICLP)*, pages 222–238, Budapest, June 1993. MIT Press.

[EHT95] K. Eder, S. Hölldobler, and M. Thielscher. A resource-oriented deductive Approach towards Hierarchical Planning. (Submitted), 1995.

[FHN72] R. Fikes, P. Hart, and N. J. Nilsson. Learning and executing generalized robot plans. *Artificial Intelligence Journal*, 3(4):251–288, 1972.

[GHS+92] G. Große, S. Hölldobler, J. Schneeberger, U. Sigmund, and M. Thielscher. Equational logic programming, actions, and change. In K. Apt, editor, *Proceedings of the International Joint Conference and Symposium on Logic Programming (IJCSLP)*, pages 177–191. MIT Press, 1992.

[GHS95] G. Große, S. Hölldobler, and J. Schneeberger. Linear Deductive Planning. *Logic and Computation*, 1995. (To appear).

[Gin93] M. L. Ginsberg. *Essentials of Artificial Intelligenze.* Morgan Kaufmann, 1993.

[Gir87] J. Y. Girard. Linear logic. *Journal of Theoretical Computer Science*, 50(1):1 – 102, 1987.

[GL93] M. Gelfond and V. Lifschitz. Representing Action and Change by Logic Programs. *Journal of Logic Programming*, 17:301–321, 1993.

[Höl89] S. Hölldobler. *Foundations of Equational Logic Programming*, volume 353 of *Lecture Notes in Artificial Intelligence.* Springer, 1989.

[Höl92] S. Hölldobler. On deductive planning and the frame problem. In A. Voronkov, editor, *Proceedings of the Conference on Logic Programming and Automated Reasoning*, pages 13–29. Springer, LNCS, 1992.

[HS90] S. Hölldobler and J. Schneeberger. A new deductive approach to planning. *New Generation Computing*, 8:225–244, 1990. A short version appeared in the Proceedings of the German Workshop on Artificial Intelligence, Informatik Fachberichte *216*, pages 63-73, 1989.

[HT93] S. Hölldobler and M. Thielscher. Actions and Specificity. In D. Miller, editor, *Proceedings of the International Logic Programming Symposium (ILPS)*, pages 164–180, Vancouver, Canada, October 1993. MIT Press.

[HT95a] S. Hölldobler and M. Thielscher. Computing Change and Specificity with Equational Logic Programs. *Annals of Mathematics and Artificial Intelligence*, special issue on Processing of Declarative Knowledge, 1995.

[HT95b] S. Hölldobler and M. Thielscher. Properties vs. Resources—Solving Simple Frame Problems. 1995. (Submitted).

[Kow79] R. Kowalski. *Logic for Problem Solving*, volume 7 of *Artificial Intelligence.* North Holland, New York/Oxford, 1979.

[Lif86] V. Lifschitz. On the semantics of strips. In *Proceedings of the Workshop on Reasoning about Actions and Plans*. Morgan Kaufmann, 1986.

[Llo87] J. W. Lloyd. *Foundations of Logic Programming*. Series Symbolic Computation. Springer, second, extended edition, 1987.

[McC63] J. McCarthy. Situations and actions and causal laws. Stanford Artificial Intelligence Project: Memo 2, 1963.

[McC86] J. McCarthy. Applications of circumscription to formalizing common-sense knowledge. *Artificial Intelligence Journal*, 28:89–116, 1986.

[McC90] J. McCarthy. Circumscription - a form of non-monotonic reasoning. *Artificial Intelligence Journal*, 13:27–39, 1990.

[Men79] E. Mendelson. *Introduction to Mathematical Logic*. Van Nostrand, Princeton, N.J., second edition, 1979.

[MH69] J. McCarthy and P. J. Hayes. Some philosophical problems from the standpoint of Artificial Intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence 4*, pages 463 – 502. Edinburgh University Press, 1969.

[Mil89] R. Milner. *Communication and Concurrency*. International Series in Computer Science. Prentice Hall, 1989.

[MM82] A. Martelli and U. Montanari. An efficient unification algorithm. *ACM Transactions on Programming Languages and Systems*, 4:258–282, 1982.

[MTV90] M. Masseron, C. Tollu, and J. Vauzielles. Generating plans in linear logic. In *Foundations of Software Technology and Theoretical Computer Science*, pages 63–75. Springer, LNCS *472*, 1990.

[Rei91] R. Reiter. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In V. Lifschitz, editor, *Artificial Intelligence and Mathematical Theory of Computation — Papers in Honor of John McCarthy*, pages 359–380. Academic Press, 1991.

[Rob67] J. A. Robinson. A review on automatic theorem proving. In *Annual Symposia in Aplied Mathematics XIX*, pages 1–18. American Mathematical Society, 1967.

[Sac73] E. D. Sacerdoti. Planning in a hierarchy of abstraction spaces. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 412–422, 1973.

[San94] E. Sandewall. *Features and Fluents*. Oxford University Press, 1994.

[Sch77] H. W. Schürmann. *Theorie- und Modellbildung*. Wiesbaden, 1977.

[She92] J.C̃. Shepherdson. SLDNF-Resolution with Equality. *Journal of Automated Reasoning*, 8:297–306, 1992.

[Thi94a] M. Thielscher. An Analysis of Systematic Approaches to Reasoning about Actions and Change. In P. Jorrand, editor, *International Conference on Artificial Intelligence: Methodology, Systems, Applications (AIMSA)*, Sofia, Bulgaria, September 1994. World Scientific Publishing Co. Singapore.

[Thi94b] M. Thielscher. *Automatisiertes Schließen über Kausalbeziehungen mit SLDENF-Resolution.* PhD thesis, Technische Hochschule Darmstadt, 1994. (in German).

[Thi94c] M. Thielscher. Representing Actions in Equational Logic Programming. In P. Van Hentenryck, editor, *Proceedings of the International Conference on Logic Programming (ICLP)*, pages 207–224, Santa Margherita Ligure, Italy, June 1994. MIT Press.

[Thi95a] M. Thielscher. Computing Ramifications by Postprocessing. In C. S. Mellish, editor, *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, Montreal, Canada, August 1995. Morgan Kaufmann. (To appear).

[Thi95b] M. Thielscher. On the Completeness of SLDENF-Resolution. *Journal of Automated Reasoning*, 1995. (To appear).

[War83] D. H. D. Warren. An abstract Prolog instruction set. Technical Report 306, SRI International, 1983.