Reducing Planning Problems by Path Reduction

Dissertationsschrift
vorgelegt am Fachbereich Informatik
der Technischen Universität Darmstadt
von
Diplom-Informatiker Ulrich Scholz

zur Erlangung des akademischen Grades eines
Doktors der Naturwissenschaften (Dr. rer. nat.)

Darmstadt 2003
Hochschulkennziffer D 17

Erstreferent: Prof. Dr. Wolfgang Bibel
Korreferentin: Prof. Dr. Susanne Biundo

Tag der Einreichung: 11. Dezember 2003
Tag der Disputation: 9. Februar 2004

Geleitwort

Planen gehört zu den charakteristischen Merkmalen von Intelligenz und ist deshalb von Anbeginn der Intellektikforschung eines ihrer zentralen Forschungsthemen. Dabei geht es der Intellektik immer um die Entwicklung eines generellen Verfahrens, also nicht um spezielle Algorithmen für jede spezifische Problemstellung, und dies in Analogie zum Menschen, der ja auch eine recht allgemeine Fähigkeit zum rationalen Schließen und Planen aufzuweisen scheint.

Inzwischen hat eine bestimmte Klasse von allgemeinen Planungssystemen einen erstaunlichen Reifegrad erreicht. Diese finden daher nunmehr zunehmend in verschiedensten Anwendungsfeldern Verwendung, beispielsweise in der Verarbeitung (vor allem bei der Generierung) natürlicher Sprache, in logistischen Aufgabenstellungen, ja selbst in Aufzugsystemen großer Hochhäuser, um nur einige der Anwendungen anzudeuten.

Ein Grund für den Erfolg dieser Systeme liegt in ihrer Beschränkung bei der Modellierung von Planungsaufgaben. Erstens wird jede Form des inferenziellen Schließens außer Acht gelassen; entsprechendes Wissen muss vom Planungsingenieur bei der Modellierung vorgeleistet werden. Infolgedessen konzentriert sich, zweitens, die Modellierung nach dem Vorbild eines der frühesten Planungssysteme, STRIPS (Stanford Research Institute Planning System), ausschließlich auf die (sich aufgrund von Handlungen oder Ereignissen ergebenden) Veränderungen. Dabei gehen diese Systeme von einer Repräsentation der Weltzustände in Form von Mengen aussagenlogischer Grundatome aus. Handlungen, die von einem zum nächsten Zustand führen, werden daher als Manipulationen an derartigen Mengen (Auftretensprüfung, Entfernung, Hinzufügung) definiert. Die Aufgabe, von einem vorgegebenen Anfangszustand zu einem Zielzustand mittels so definierter Handlungsoperatoren zu gelangen, ist in dieser Formulierung konzeptuell relativ einfach. Gleichwohl handelt es sich komputational um ein sehr schweres Problem (komplexitätstheoretisch PSPACE-vollständig).

Wie bei anderen komplexitätsmäßig schweren Problemen (etwa in der Deduktion) lohnt es sich für die Praxis gleichwohl, einen allgemeinen und exponentiellen Algorithmus mit Maßnahmen zu flankieren, die das gegebene Problem in einem Vorverarbeitungsschritt aufgrund seiner speziellen Struktur so weit wie möglich reduzieren, so dass die dann noch verbleibende Arbeit des allgemeinen Algorithmus auf ein Minimum beschränkt ist. Erfolgreiche Planungssysteme setzen hier eine Vielfalt von Vorbereitungsschritten ein. Eine theoretische Untermauerung und damit einhergehende Verfeinerung und Verbesserung der verwendeten Techniken wurde auf diesem Spezialgebiet bislang jedoch nicht geleistet. Diese Lücke versucht Herr Scholz mit seiner hier als Buch vorliegenden Dissertation zu schließen.

Um eine Vorstellung vom Charakter der Analyse zu vermitteln, sei einer der zentralen Begriffe der Arbeit, nämlich der sogenannten c-Invarianten kurz illustriert. Hier handelt es sich um eine Menge von Grundatomen, die sich in jedem denkbaren Zustand gegenseitig ausschließen, wie beispielsweise der Deckel meines Laptops entweder geschlossen oder offen ist, wie immer auch sonst die Welt um ihn herum beschaffen sein mag. Enthält sowohl der Anfangs- wie auch der Endzustand eines Planungssystems je ein Element dieser c-Invariante, so kann man diese beiden Elemente mit der entsprechenden Folge von Handlungen (wie zB. dem Schließen des Deckels) ineinander überführen. Eine solche Folge wird ein Pfad genannt, dem zweiten wichtigen Begriff der Arbeit. Irgendwie muss diese Folge Bestandteil des gesamten Lösungsplanes sein, der darüber hinaus noch weitere Manipulationen an den Grundatomen außerhalb der betreffenden c-Invariante vornimmt, um den gesamten Endzustand herzustellen. Die Grundidee der Arbeit besteht nun darin, bei einer gegebenen Planungsaufgabe irrelevante Pfade – wie beispielsweise Zyklen von Handlungen – so weit wie möglich zu eliminieren und den Lösungsplan durch Kombination von Pfaden aus der Restmenge zusammen zu fügen.

So einfach diese Idee ist, in komplexeren Planungsproblemen lässt sie sich leider nur sehr eingeschränkt realisieren. Denn die Pfadhandlungen zu einer c-Invariante können natürlich mit denen einer anderen gekoppelt sein. In aller Allgemeinheit sind dem Erfolg solcher Überlegungen also schnell wieder komplexitätsmäßige Grenzen gesetzt. Gleichwohl ist eine solche Analyse im Hinblick auf Problemdomänen mit speziellen Strukturen sehr aussichtsreich. Im vorliegenden Buch wird dieser Schritt hin zu speziellen Domänen jedoch späteren Untersuchungen und der Praxis überlassen. Es liefert dafür die allgemeine theoretische Basis.

Ich wünsche dem Buch Erfolg im Sinne von noch besseren Planungssystemen, die in dem unbegrenzten Anwendungsfeld des Planens erfolgreich sind und der Intellektik weiter zu dem ihr gebührenden Ansehen unter den Wissenschaften verhelfen.

Günterfürst, den 9. Februar 2004

Wolfgang Bibel

Danksagung

Ein langer Weg hat seinen Abschluß gefunden; nun ist es Zeit, innezuhalten und zurückzuschauen um sich der beschrittenen Pfade und der treuen Begleiter zu erinnern.

Zuallererst möchte ich meinem Doktorvater Prof. Bibel danken. Er hat mir immer Freiraum für meinen eigenen Weg gegeben und mir gleichzeitig vermittelt, diesen Weg im Zusammenhang mit den großen Fragen der KI zu sehen. Desweiteren gebührt mein Dank Prof. Biundo, der Zweitgutachterin dieser Arbeit. Frau Biundo hat mir, als Koordinatorin des European Network of Excellence in AI Planning (PLANET), die Möglichkeit gegeben, Orte zu besuchen und Menschen zu treffen, ohne die diese Arbeit nicht so wäre, wie sie heute ist.

Besonders ist mir der Besuch des KPLABs in Linköping, Schweden in Erinnerung, der von PLANET als Cross Site Visit unterstützt wurde. Während dieses Aufenthalts haben die Ideen der vorliegenden Arbeit angefangen Gestalt anzunehmen. Mein Dank gilt hier meinem Kollegen Patrik Haslum, mit dem mich eine anregende Zusammenarbeit verbindet, deren Ergebnisse in diese Arbeit eingeflossen sind.

Anteil am Entstehen dieser Arbeit hatten außerdem Stephan Schulz, der mir mit Rat und Tat für seinen Beweiser E zur Seite stand, Joachim Schimpf, der mir so manche subtile Besonderheit des ECL$^{\text{i}}$PS$^{\text{e}}$ Systems erklärte, und Georg Fette, dessen DKEL Parser im erstellten System verwendet wird. Gunter Grieser, Peter Grigoriev, Hesham Khalil und Thomas Stützle haben Teile dieser Arbeit gelesen und mit Hinweisen und konstruktiver Kritik zu ihrer Fertigstellung beigetragen. Joachim Nadler hatte immer ein Ohr auch für die abwegigsten Fragen und lenkte mich mit seinem Rat auf sicheren Grund zurück.

Befördert wurde meine Arbeit durch die angenehme Atmosphäre unter meinen Kollegen des Fachgebiets Intellektik und des befreundeten Fachgebiets Programmiermethodik. Insbesondere möchte ich hier Klaus Varrentrapp erwähnen, mit dem mich die Zusammenarbeit an einer Testumgebung für Planungssysteme verbindet, sowie Thomas Stützle und Michael Thielscher, die mich für das Thema Planen begeisterten und die mich auch später immer begleitet haben. Maria Tiedemann danke ich – für einfach alles.

Außerhalb Darmstadts erinnere ich mich an interessante und fruchtbare Diskussionen mit Stefan Edelkamp, Maria Fox, Malte Helmert, Jörg Hoffmann, Derek Long, Lee McCluskey, Fabrizio Morbini, Ioannis Refanidis, Jussi Rintanen und vielen anderen.

Am meisten Dank gebührt meinen Eltern und Freunden. Nichts wäre ich ohne ihre Unterstützung, sie gaben mir Halt in einer – auf für sie – nicht einfachen Zeit.

Contents

List of Figures

List of Examples

Chapter 1

Introduction

1.1 Motivation

It is an old dream to build machines that have the same capabilities and show the same behavior as humans. The advent of computers suggested embedded programs as tools for this task and the corresponding field of computer science, known as Intellectics or Artificial Intelligence, appeared shortly thereafter. The goal of Intellectics is the development of computer programs whose behavior is considered intelligent if it is exhibited by humans. Hereby, intelligence is not restricted to complex problems and puzzles but comprises all aspects of human behavior, including the task of baking a cake. Let us examine this example more closely.

Imagine we stand in our kitchen and have the desire to eat a cake. Because our pantry is well filled with all necessary ingredients we decide to bake it on our own. We then simply grab a cook book and start the preparations. But what if we do not have any cook book at hand? Even without detailed instructions, our knowledge of food preparation allows us to bake a cake, though the task becomes harder than using a recipe. First we step back and examine the situation. We identify simple steps and their dependencies and search for additional information that helps us with our task, i. e., aspects of the world that are relevant for our problem. Based on these findings we then form a plan that eventually results in a cake. Once we successfully executed the plan we can eat the cake and, if we are satisfied with the outcome, we add it as a new recipe to our collection.

The problem of finding a recipe for baking a cake is an instance of a planning problem. In short, planning is the task of finding a plan, i. e., a sequence of actions whose execution transfers a given world state in another one that satisfies a certain goal. Without doubt, planning constitutes a central aspect of human intelligence and we are constantly planning in our daily life. Similarly, planning is a core process of intelligent agents and this form of general problem solving is a major area of Intellectics from the beginnings of the field.

How can a computer solve our cake baking problem? It first needs a representation of our kitchen; let us call it the kitchen world. The kitchen world formalizes all objects in the world that are related to baking a cake, for example some eggs, an oven, and the frying pan. In addition it formalizes actions that correspond to the simple steps of food preparation that we considered in our example. After providing an initial situation of the kitchen world, i. e., the filling of our pantry and the cake as desired goal we can search through this representation for a plan.

Surprisingly, planning is a hard problem for computers although many planning problems appear to humans almost trivial. For a long time the capability of planning systems was restricted to toy problems. The subsequent improvement of planner performance is in part due to the development of techniques that correspond to what we have described above as "stepping back and examining the problem": Planners perform *domain analysis* prior to the search for a plan, i. e., they apply simple techniques to identify relevant parts of the problem domain, e. g., the frying pan, and to find useful laws that hold during the execution of a plan. An example for the latter is the uniqueness of the location of physical objects. Such knowledge is then used in various ways, depending on the specific technology of the planning system. It is fair to say that the development of new domain analysis techniques is important for further improvement of planning performance.

1.2 Research Motives and Contributions

As we just pointed out, domain analysis is an important part in the task of planning. In this thesis we present a particular domain analysis technique called *path reduction*. It exploits a structure called *c-invariants*, which is commonly present in a planning problem. The casual reader is referred to Section 3.1.1 on Page 16 for an informal introduction to the concept of c-invariants and to related concepts. If a planning domain is structured by c-invariants then path reduction allows to extract irrelevant information from problems in this domain. Our design of path reduction guarantees completeness and optimality, i. e., if a problem is solvable then the application of path reduction preserves an optimal solution. Furthermore, it is applicable to general planning problems, i. e., it does not pose syntactic restrictions on the input. Altogether our approach yields the following contributions.

c-Invariants

The performance of many current planning systems is based on the knowledge and exploitation of state invariants, of which c-invariants are a common subclass. We study c-invariants in detail and analyze the properties of planning domains that result in the existence of c-invariants. Furthermore, we examine the interaction between different c-invariants and between a c-invariant and the part of a planning domain that is not structured by c-invariants.

Path Replacement

c-Invariants divide a planning problem into subproblems and a plan can be considered as the combination of paths, i. e., solutions for these subproblems. Paths for a given subproblem are similar across different solutions to the planning problem. We use this similarity of paths to develop a notion of general path replaceability.

Path Reduction

Another contribution of this thesis is the development of the technique of path reduction. We use path replaceability to find sets of paths, whose actions are sufficient to find a solution to the problem, so that other actions can be excluded from the problem.

Implementation

Our implementation of path reduction uses a non-ground representation and is applicable to planning problems without syntactic restrictions. To combine these two design goals we use deductive techniques, i. e., theorem proving. Our implementation is based on a constraint solver which uses a theorem prover as subsystem, thus minimizing the use of the latter.

1.3 Outline of the Thesis

The thesis is organized as follows.

Chapter 2 outlines the main concepts and the background of the work presented in this thesis. We introduce STRIPS planning and give a short overview of planning techniques. Furthermore, we outline the concept of domain analysis and motivate our focus on general and lifted domain analysis techniques.

Chapter 3 introduces two kinds of domain knowledge: c-invariants and replaceable sequences of actions. In particular, it discusses various aspects of c-invariants, the structures of planning domains that give rise to c-invariants, and the interactions between c-invariants during the execution of plans.

Chapter 4 develops path replaceability.

Whether a path is replaceable does not tell us whether it is necessary. In Chapter 5 we develop path reduction, a constructive technique to find sets of paths from a planning problem that are sufficient to find a solution to that problem.

In Chapter 6 we describe the implementation of path reduction and evaluate its feasibility.

Finally, in Chapter 7, we summarize the main contributions of this thesis, describe related work, and outline directions for further research.

Chapter 2

Planning

In this chapter we introduce STRIPS planning and give an overview of planning techniques. We then motivate the use of explicit domain knowledge and its derivation from unrestricted, lifted domains.

2.1 STRIPS Planning

In the following, we introduce STRIPS style planning [16]. STRIPS is a simple yet powerful planning formalism. It has been developed for the *ST*anford *R*esearch *I*nstitute *P*roblem *S*olver, one of the first practical planning systems and is still the base for the majority of contemporary planning systems.

A formalization in STRIPS describes a world by a finite set of propositional properties, which we call facts. Each fact is modeled by an atom that can be true or false at a given time. For example, the fact *have*(*eggs*) of our kitchen world models the property of having eggs in the pantry. If it is true then we have eggs, if it is false then not. The configuration of the world at a certain time is modeled as *state*. A STRIPS state is the set of all facts that are true at a given time.[1] Hereby, we implicitly use the closed world assumption, i. e., we assume the negation of all facts that are not in a state. In other words, a fact is false in a state if it is not element of that state.

Properties of the world that do not correspond to a fact are assumed to be irrelevant, e. g., whether the eggs are brown or white matters only if the kitchen world comprises a fact that denotes their color. Consequently, we can distinguish two configurations of the world in STRIPS only in respect to those properties that are modeled as facts; configurations that solely differ in other aspects correspond to the same state and are indistinguishable.

Definition 2.1 (Facts and States)
A fact *is an atom and a* plan state*, also simply called* state*, is a finite set of facts. We say that a fact f is* true *in a state s if $f \in s$. Otherwise, we say that f is* false *in s. We also use* active *as synonym for true.*

We denote states by the letter s and facts by the letter f. Sometimes, we also use g and h for facts. ◇

[1] To be precise, we introduce the style of planning that is referred to as STRIPS planning nowadays. States of the original STRIPS formalism comprise first order formulas.

Usually, we need to change the world in order to eat a cake. We model the ability to alter the world by *actions*: As stirring the eggs and flower makes a dough in the real world, we define the action *stir* to change a state s_1 in which the facts *have*(*eggs*) and *have*(*flower*) are active to a state s_2 in which we have a dough. After stirring, the flower and eggs are gone, so $\{have(eggs), have(flower)\} \cap s_2 = \emptyset$. In other words, the action removes the facts *have*(*eggs*) and *have*(*flower*) from a state and includes the fact *have*(*dough*); we say that *stir deletes* the two former facts and *adds* the latter. The combined added and deleted facts of an action are also called the *effects* of that action.

Definition 2.2 (Actions)
An action *o is a triple* $\big(pre(o), add(o), del(o)\big)$ *where each component is a finite set of facts representing its* preconditions*, its* added*, and its* deleted effects*, respectively. Furthermore, we require action o to satisfy* $del(o) \cap add(o) = \emptyset$.

As abbreviations, we define the consumed facts *of o as* $cons(o) = pre(o) \cap del(o)$ *and the* postconditions *of o as* $post(o) = \big(pre(o) \setminus del(o)\big) \cup add(o)$*. We denote actions by the letter o to avoid confusion with the common article 'a'.* ◇

The condition that the added and the deleted effects of an action have no common fact prevents anomalous actions. Before we motivate this restriction, as well as the definition of postcondition, we first explain the role of preconditions and introduce the execution of actions.

The triple $\big(\{have(gloves), in(cake, oven)\}, \{have(cake)\}, \{in(cake, oven)\}\big)$ is the action *take-out*(*cake*, *oven*) of our kitchen world, which models the removal of the cake from the oven. Obviously, the action is futile without a cake in the oven. Furthermore, the oven is hot after baking the cake, so that we better have gloves while opening it. In other words, the given action is reasonable only in some states of the world but not in all: If *have*(*gloves*) is false then the cake will stay in the oven. Such dependencies of an action on a state is formalized by its preconditions as a third set of facts beside the effects and we say that the action is *applicable* in a state if its precondition is a subset of that state.

The ultimate motivation for an action is its *execution*, i. e., the resulting change of the world. STRIPS models the execution of an action in a state by removing its deleted facts and including its added facts. For example, the execution of *take-out*(*cake*, *oven*) consumes the fact *in*(*cake*, *oven*) and adds the fact *have*(*cake*). Removing the cake from the oven does not consume the gloves, so they are still present afterwards. Consequently, *have*(*gloves*) is a postcondition of *take-out*(*cake*, *oven*), although it is not added by this action.

Definition 2.3 (Applicability and Execution of Actions)
An action o is applicable *in a state s if* $pre(o) \subseteq s$*. If o is applicable in s then the* execution *of o in s yields the state* $\big(s \setminus del(o)\big) \cup add(o)$*. If o is not applicable in s then the result of executing o in s is undefined. Sometimes we use* application *as synonym for execution.* ◇

If an action is not applicable in a state then the result of its execution is undefined. To prevent such anomalous states we stipulate that we execute an action in a state only if it is applicable in that state. The condition $add(o) \cap del(o) = \emptyset$ has a similar motivation: If an action o would add and delete the same fact f then the truth of f would be undefined after the execution of o. As a consequence, we can consider each effect of an action independent

of the others and of the context, and the postconditions of an action are exactly those facts that are known to be true after the application of that action.

Note that our definition of STRIPS makes an implicit assumption about the nature of actions, which is called the *STRIPS assumption* [67]. It states that actions can always be correctly and completely described by a triple of fact sets. In general, we assume that the execution of an action affects only the facts mentioned in its effects; all other facts remain unchanged. In particular, an action has no hidden properties that interferes with its applicability and it has no indirect effects. In this way STRIPS solves the so-called *frame problem*, i. e., the problem of persistence: If a fact is not mentioned in the effects of an action then it persists during the execution of that action.

We now know how to execute single actions; is this sufficient to bake a cake? Let us reconsider the motivational example: Our goal is to have a cake, i. e., to reach a state with $\mathit{have}(\mathit{cake})$ active. We know that this fact is added by the action $\mathit{take\text{-}out}(\mathit{cake}, \mathit{oven})$. If this action is applicable in our current state, i. e., if $\mathit{in}(\mathit{cake}, \mathit{oven})$ is true then we have a cake after its execution. But what if not? Well, then we have to reach a state in which $\mathit{in}(\mathit{cake}, \mathit{oven})$ is true. In other words, the fact $\mathit{in}(\mathit{cake}, \mathit{oven})$ is a new goal, similar to our initial one, and we reach our initial goal by first executing an action to reach a state with $\mathit{in}(\mathit{cake}, \mathit{oven})$ active, followed by an execution of $\mathit{take\text{-}out}(\mathit{cake}, \mathit{oven})$, i. e., by consecutive execution of a sequence of actions. Such a sequence we a call *plan*.

Definition 2.4 (Sequences and Plans)
Let $\Sigma = \{\dots, o_i, \dots\}$ be a set. Then a sequence *$\langle o_1, o_2, \dots \rangle$ is an element of Σ^*. We write $\langle\,\rangle$ for the empty sequence and $T \circ T'$ to denote the* concatenation *of the sequences T and T'. In the following, Σ will always be a set of actions.*

A sequence $\langle o_1, \dots, o_n \rangle$ of actions, also called action sequence *or simply* sequence*, is* applicable *in a state s if o_1 is applicable in s and for each $i \geq 2$, o_i is applicable in the state resulting from applying $\langle o_1, \dots, o_{i-1} \rangle$ in s. A sequence is* conflict-free *if there is a state in which it can be applied, and a* plan *is a conflict-free sequence. If a sequence is not conflict-free then we say it has a* conflict. ◇

The subject of planning is the synthesis of a solution plan for a given planning problem. Hereby, a problem consists of a set of available actions, the *initial state* of the world, and a set of facts called the *goal*. A plan is a *solution* of a planning problem if its execution in the initial state yields a state that is a superset of the goal.

Definition 2.5 (STRIPS Planning Problems)
A planning domain*, or* domain *for short, is a set of STRIPS actions. A* planning problem*, or* problem *for short, Ψ is a triple $(\mathcal{O}, \mathcal{I}, \mathcal{G})$, where $\mathcal{O}$ is a domain, $\mathcal{I}$ is a state, and $\mathcal{G}$ is a set of facts. We call $\mathcal{I}$ the* initial state *and $\mathcal{G}$ the* goal *of the problem. A* solution *to Ψ is a plan S such that all actions of S are included in $\mathcal{O}$, S is applicable in $\mathcal{I}$, and each fact $f \in \mathcal{G}$ holds in the state resulting from executing S in $\mathcal{I}$.*

Usually, we write T for action sequences and plans, and S for solutions. We use $\mathcal{F}$ to denote fact sets. We write $\mathcal{F}_\Psi$ for the set of all facts of a planning problem Ψ, i. e., all facts that occur in an action of Ψ, in the initial state, or in the goal. ◇

We now have three terms that refer to a list of actions: action sequence, plan, and solution. As a solution is a plan and a plan is a sequence, we sometimes have the choice of how to reference a sequence. We then choose the term that indicates its intended role.

Plans differ from action sequences in that the former are conflict-free and the latter can be arbitrary, i. e., can have a conflict. We sometimes call a plan a sequence to emphasize that it is not meant to be executed on its own, e. g., that it is a part of a larger plan. A solution is a plan that solves a given planning problem. More generally, the term solution points to the state before and after its execution. If we want to underscore this property of a plan then we sometimes call it a solution; calling it a plan instead would mainly emphasize its conflict-freeness.

In this work we often compare sequences and parts thereof. We therefore define the following notations.

Definition 2.6 (About Sequences)
Let $T = \langle \ldots, o, \ldots \rangle$ be a sequence. Then we say that o is an element in *T or simply that o is* in *T. A sequence T' is a* partial sequence *of T if the elements of T' are an ordered choice of elements of T. We call T a* proper *partial sequence of T' if it differs from T.*

A subsequence *of T is a continuous partial sequence of T. A* prefix *(*postfix*) of T is a subsequence of T that begins with the first (ends with the last) element of T, respectively. An* infix *of T is a subsequence of T that is neither a prefix nor a postfix.*

Let T' be a subsequence of a sequence T. We say o lies during *T' in T if o is in the shortest partial sequence of T that has T' as subsequence.* ◇

Finally, we extend the notions of preconditions, effects, and postconditions to plans.

Definition 2.7 (Preconditions and Effects of a Plan)
Let T be a plan and $T = T' \circ \langle o \rangle$. Then the preconditions *of T, the* added *and* deleted effects *of T, and the* postconditions *of T are inductively defined as*

- $\mathit{pre}(T) \equiv \mathit{pre}(T') \cup \big(\mathit{pre}(o) \setminus \mathit{add}(T')\big)$,
- $\mathit{add}(T) \equiv \big(\mathit{add}(T') \setminus \mathit{del}(o)\big) \cup \mathit{add}(o)$,
- $\mathit{del}(T) \equiv \big(\mathit{del}(T') \setminus \mathit{add}(o)\big) \cup \mathit{del}(o)$, *and*
- $\mathit{post}(T) \equiv \big(\mathit{post}(T') \cup \mathit{pre}(o) \cup \mathit{add}(o)\big) \setminus \mathit{del}(o)$,

respectively, where each of $\mathit{pre}\big(\langle\,\rangle\big)$, $\mathit{add}\big(\langle\,\rangle\big)$, $\mathit{del}\big(\langle\,\rangle\big)$, and $\mathit{post}\big(\langle\,\rangle\big)$ is the empty set. ◇

The recursive nature of these definitions allows an easy verification that a plan T is executable in a state s if $\mathit{pre}(T) \subseteq s$ and that the resulting state of its execution is $\big(s \setminus \mathit{del}(T)\big) \cup \mathit{add}(T)$. The set $\mathit{post}(T)$ comprises all the facts that are guaranteed to be true after the execution of T and the execution of T always yields a state s' with $\mathit{post}(T) \subseteq s'$.

Let us exemplify the planning terminology with a formalization of our kitchen world before we continue with an overview of planning techniques.

Example 2.1 (The Kitchen World)
The kitchen world is a (highly abstracted) formalization of baking a cake. It comprises the following action schemata: stir, put-into(dough, oven), use-oven, and take-out(cake, oven). The first action corresponds to stirring the ingredients of the cake, in our case just flower and eggs, to get a dough. The

remaining actions put the dough in the oven, bake the cake, and remove it from the oven, respectively. The following table gives their full definition.

action	*preconditions*	*added facts*	*deleted facts*
stir	*have(flower), have(eggs)*	*have(dough)*	*have(flower), have(eggs)*
put-into(dough, oven)	*have(dough)*	*in(dough, oven)*	*have(dough)*
use-oven	*in(dough, oven)*	*in(cake, oven)*	*in(dough, oven)*
take-out(cake, oven)	*have(gloves), in(cake, oven)*	*have(cake)*	*in(cake, oven)*

The initial state $\mathcal{I}$ of the cake planning problem Ψ is the set $\{have(flower), have(eggs), have(gloves)\}$; the goal $\mathcal{G}$ is the set $\{have(cake)\}$. In addition, the set $\mathcal{F}_\Psi$ comprises the facts have(dough) and in(dough, oven). The solution is, of course, the sequence $S = \langle stir, put\text{-}into(dough, oven), use\text{-}oven, take\text{-}out(cake, oven)\rangle$. We verify the solution by computing $pre(S) = \mathcal{I}$ and $post(S) = \{have(cake), have(gloves)\}$. In other words, S is applicable in the initial state and its execution yields a state in which we have the cake. ■

2.2 Planning Techniques

The motivation for the advance of planning techniques can be divided into two directions: the expressiveness of the planning formalism, i. e., the adequateness of the underlying world model, and the performance of the solution process. Usually, planners are tailored to a specific formalism, so that a change in expressiveness requires to extend, and in most cases to change, the applied planning technology. In this work we develop planning technology for STRIPS and its extension to other formalisms is beyond its scope.

The second direction of planning research is concerned with the performance of the solution process, i. e., with techniques to handle larger problems, with the quality of solutions, and with the solution speed, to name just a few. Various techniques have been proposed to achieve these ends since the beginning of the field, which is in part a result of an apparent discrepancy: We encounter numerous planning problems in our daily life and most are easy for us. On the other hand, planning is computationally hard. In its STRIPS variant, planning is *PSPACE*-complete[2] and planning is still *NP*-complete even with severe restrictions on the considered planning domains [11]. We now give an overview of the development of planning technology in respect to performance.

Planning is the synthesis of a solution for a given planning problem. Simply said, it consists of building a search space that comprises solutions to a given problem and of searching that space until a solution is found. In principle, planners can use any representation of planning problems and of plans if they can search it for a solution – and numerous alternatives have been explored.

Early planning systems generally search in the space of totally ordered action sequences. They find action sequences for solving each goal fact separately and combine them to a final solution. This so-called *linear planning* [59] is not *complete*, i. e., it does

[2]We assume that the reader is familiar with the field of computational complexity. An introduction into this topic can be found in [23].

not guarantee to yield a solution to a solvable problem. Furthermore, such a simple technique has to consider many irrelevant plans, e. g., plans that are unrelated to the goal of the problem, sets of plans that only differ in the ordering of unrelated actions, and so on. Consequently, the performance of these planners is low.

The subsequent work on planning aims at complete planners. One solution is *regression planning* [67,69], which reorders the actions in the combined sequence to eliminate conflicts. Another idea is *partial-order planning* [59,64]. A partial-order planner examines causal relationships between actions, i. e., it detects conflicts between actions in a plan and orders them accordingly. Simply said, actions in a partially ordered plan are ordered only if their order makes a difference and their order is left unspecified otherwise. Consequently, similar totally ordered plans are grouped into equivalence classes and the search space is comprised by their representatives. Partial-order planning is significantly more effective than simple total-order planning, as for example demonstrated by the planner SNLP [46,63]. This advance is due to the combination of a small search space with an efficient way of enumerating candidate solutions.

Despite its improvements, partial-order planning is still too slow to solve problems of realistic size.[3] The low performance is in part due to plan candidates that are considered by partial-order planners although humans would discard them right away. The reasons are simple properties of planning domains whose violation immediately rule out a candidate, e. g., the uniqueness of object locations: If an action sequence requires an object to be at two places simultaneously then it is not part of a solution. Simple partial-order planner have to discover such laws through exhaustive search, and because they have no memory they have to rediscover it over and over again. Consequently, they spend a lot of time in areas of the search space that do not contain solutions.

Partial-order planning dominated the planning research until the advent of planners that are based on efficient encodings of the search space. A first example is *planning by satisfiability* [37]. Planners as SatPlan [37] and BlackBox [38] encode a planning problem as SAT formula and use a generic SAT solver to find a satisfying assignment. Hereby, the encoding guarantees that a solution to the formula corresponds to a solution to the initial planning problem. The success of SAT spurred the search for further ways to encode planning problems. Examples include encodings into ordered binary decision diagrams [15] and into constraint satisfaction problems [27].

The success of planning by satisfiability was in part due to the vast improvements in the field of SAT solvers. Another reason is their use of additional knowledge: The performance of planning by satisfiability is significantly improved if the formula is enriched by knowledge about the problem, e. g., the uniqueness of object locations, even if this knowledge is embodied by the initial SAT formula. The observation of simple properties and the impact of their use have led to techniques that reduce the search space beforehand. Here, the planner employs a search for simple but useful information about the problem prior to the search for a solution plan. This so-called *domain knowledge* is then used to reduce the planning problem to another one, in which unpromising areas of the search space are not present.

The next improvement in planning technology was the planning graph as introduced by the planner Graphplan [9]. The planning graph is a compact representation of all

[3] Recently, this common impression has been challenged (cf. [51]).

plan prefixes of a certain length, enriched by causal relationships and domain knowledge. Graphplan builds its planning graph incrementally and uses exhaustive searches either to find a solution or to prove that there is no solution of the considered length. In the latter case, it extends the graph and recurses. During the construction of the graph, Graphplan computes knowledge that is similar to the additional knowledge used by SAT based planners and employs it to cut dead branches of the search. Several successors of Graphplan, e. g., IPP [42], STAN [43], and LPG [27] to name just a few, build on the planning graph by adding more knowledge and by improving its representation.

A recent development of planning technology is the use of search heuristics, which steer the search towards promising areas of the search space. An example are greedy regression tables of the planner GRT [54], which guess the length of a plan from a given state to the goal. This heuristic is then used by a forward search in the space of plan prefixes. Another example is FF [34], whose heuristic is related to the planning graph.

A common property of almost all recent planners, beginning with the SAT based ones, is their use of domain knowledge. We now give a short introduction into this field.

2.3 Domain Analysis and Domain Knowledge

The area of *domain analysis*, a subfield of planning research, is concerned with techniques to infer knowledge from the input to a planner before the search for a solution and with the use of this knowledge during the search. In fact, almost all contemporary planners use domain analysis in some way. This section defines the resulting domain knowledge in respect to the input of planning systems and advice, and gives examples for the use of this knowledge.

We can divide the knowledge input to a planning system into two distinct classes: problem specification and advice. The problem specification in turn typically consists of two parts: a description of the means at the planners disposal, such as the possible actions that may be taken and resources that may be consumed, and the goals to be achieved, including possibly a measure that should be optimized, constraints that should never be violated, and so on. Hereby, we use the term advice for knowledge of all kinds, intended to help the planner find a better solution, find it more quickly or even to find a solution at all.

In between specification and advice, we distinguish a third class of knowledge, commonly called domain knowledge. Briefly, it consists of statements about a planning problem that are logically implied by the problem specification, but that is not part of the specification. There are several good reasons for making this distinction. First of all, domain knowledge is implied by the problem specification, so it is often possible to derive it automatically. In this way it is different from advice, which must be provided by a domain expert or learned from experience over many similar problems. In short, domain analysis is the process of making domain knowledge explicit.

As good planner advice tends to depend on knowledge both about properties of the problem and the planner, it differs from domain knowledge. Nevertheless, there is often a fairly direct mapping from certain classes of domain knowledge to useful advice for a given planner. To take a simple example, for a planner that searches the state space backwards from the goal, an obvious use of state invariants is to cut branches of the search tree that

violate an invariant. This is a sound principle, since a state that violates an invariant can never be reached and thus a goal set that violates the invariant is unreachable. The principle is founded on knowledge of how the planner works, but depends also on the existence of a certain kind of domain knowledge, namely state invariants.

On the other hand, domain knowledge in itself does not determine its use for advice. To continue the example, state invariants have many more uses: the MIPS planner uses them to find efficient state encodings [14] and to find abstractions for generating heuristics [13]; in GRT [54] they are used to split the problem into parts and to improve the heuristic. In principle, the same planner can use the same domain knowledge in different, mutually exclusive ways.

The use of explicit domain knowledge opens up the possibility of reasoning about the planning process. This use is demonstrated by the planner HAP [66], whose planning strategy is adjusted according to the existence and characteristic of domain properties. Statements of domain knowledge, e. g., a state invariant, are regarded as property of the corresponding domain, similar to details as the number of goal facts. It also allows a planner to identify subproblems which can be solved by specialized planners, as demonstrated by the planner Hybrid STAN [20].

In conclusion, the further development of domain analysis techniques will lead to better planners and a deeper understanding of the relationship between features of planning problems and planner performance.

2.4 General and Non-Ground Planning Techniques

Usually, planning problems are defined in a much more compact way than the one that we introduced to formalize planning. For example, our definition of STRIPS uses ground atoms to model properties of the world and actions are composed by sets of ground atoms. More naturally, we would think of planning by means of parameterized operators, whose ground instances correspond to sets of actions. This idea is related to the idea of lifting, which was introduced by Robinson [57] to find a proof procedure for first order logic in analogy to techniques used for propositional logic. Here, we use the term to indicate that we write and handle a parameterized planning problem without transforming it into its ground representation. In other words, a planning technique is lifted if it uses a general level representation.

Many current planning techniques, e. g., the planners Graphplan [9] and FF [34] as well as domain analysis techniques RIFO [50] and RedOp [32], are not lifted but ground. In other words, they transform their input into a ground representation by instantiating all parameters in all possible ways. The resulting representation can be stored and manipulated in a computationally efficient way. For example, the planner STAN [43] represents states as bit vectors, where each bit corresponds to a specific fact. If a bit is “1” then the corresponding fact is active in that state. As bit operations are very fast, grounding allows to implement highly optimized planning techniques.

Grounding has two apparent drawbacks. First, the size of the representation is substantially increased, although current grounding based planners use domain analysis and sophisticated instantiating techniques [41] to limit the size of the resulting representation. Nevertheless, if we want to increase the size of problems further, we will eventually have

to use lifted techniques. The second drawback is a loss of explicit structure. If actions are represented as bits then they all are alike and it is no longer clear whether two actions result from instantiating the same or different parameterized operators. Consequently, planning techniques, e. g. TIM [18], that use such information can no longer be used.

Other planning techniques are lifted but are only applicable to a class of planning problems that is syntactically restricted. Hereby, the exclusion of special cases allows to simplify the algorithms. An example is the planner IPP [42], which does not consider actions that result from unifying two parameters of an operator with the same constant. Hereby, its applicability is restricted but it never has to consider, e. g., a precondition with two equal facts. Another example is the domain analysis technique TIM [18], which solely considers literals that co-occur in the precondition and effect of operators, i. e., pairs of literals in the precondition and in the effect of an operator that have the same predicate symbol and the same parameter vector. Consequently, TIM does not consider literals that unify otherwise, even though they are part of a domain structure at which it is aimed.

Current planners which are based on grounding are capable of solving large planning problems. Likewise, tools that use syntactic restrictions yield excellent results. But both approaches are somehow unsatisfactory and are not suitable for all techniques of interest. For these reasons, it is interesting to aim for a design that is as general as possible.

A likely consequence of implementing a domain analysis technique for unrestricted, general representations is the need for deductive techniques. Consider the (easy) question whether or not a given parameterized action of a planning domain can always replace another in solution plans. This is true if for every ground instance of the parameterized action there is a corresponding ground instance of the first. Answering this question in general basically corresponds to proving a theorem. We do not propose general theorem proving as main technique to solve planning problems. Instead, we use constraint programming to find candidate statements of domain knowledge. We then employ a theorem prover as a subsystem to accept or reject these candidates. Hence, the theorems remain rather simple compared to those resulting from a full axiomatization of the domain analysis technique.

Chapter 3

Domain Analysis Techniques

This chapter introduces two kinds of domain knowledge: replaceable sequences of actions, or RSA for short, and a class of state invariants called c-invariants. In planning, both kinds of domain knowledge are well-known for their potential in reducing planning problems and improving the performance of planning systems. In the presented work, we aim at a reduction technique that combines their potential. We begin this venture with a study of the properties of planning problems that result in the existence of c-invariants. Hereby we go beyond the detail undertaken for the current use of state invariants. In the following, we analyze the structural features of planning problems that underly c-invariants and RSA, and provide the foundation for the upcoming treatment of path reduction.

3.1 c-Invariants

For every planning domain we can formulate properties that hold in every state of that domain. More precisely, such a property holds in a state of its world if it is satisfied in the preceding state. In other words, these properties are invariant regarding the execution of actions, hence their name *state invariants*. As states are sets of ground facts, state invariants, or invariants for short, are formulas over the ground facts of a planning domain.

State invariants constitute an important piece of domain knowledge. The simplest invariant, the formula "true", holds for all states of all planning domains and problems, which justifies the claim at the beginning of this section. The more specific an invariant is, the fewer states satisfy it. If a state invariant induces a tight (upper) bound on the reachable states of a planning problem, it allows to reason about classes of plans and solutions without explicitly searching them, e. g., whether there are solutions at all. Because of their ability to reduce the search space beforehand the use of state invariants is common in planning.

The existence of state invariants for a planning problem is a result of properties of actions in its domain: The most prominent is that if an invariant is true in a state then all applicable actions observe it. Consequently, the invariant is true in the succeeding state, too, and therefore in all states reachable from the first. In addition to this general property, state invariants are related to various others and we can classify invariants according to the properties which induce them.

Path reduction is based on a specific class of such state invariants, namely *c-invariants*. In the following, we first give an informal description of this class of state invariants and the properties of actions that it corresponds to; then we continue with a formal definition and detailed analysis.

3.1.1 c-Invariants – An Overview

Consider the property of unique object location, i. e., the fact that objects are exactly at one place at a time. Actions that place an object at a certain position take away the same object from another position, so that the application of actions preserves the uniqueness of object location. More generally, such a state invariant corresponds to the collection of all (internal) states in which an object can be, including its locations. In a planning domain each such internal state of an object is represented as a fact. Consequently, for each state invariant of unique object location there is a set of facts of which at most one is true in every state of the world.[1] In most of the planning domains currently under investigation, such properties of internal state correspond to c-invariants which are a class of state invariants. We identify a c-invariant with its set and stipulate that the term "set C is a c-invariant" indicates that C is a set of facts of which at most one is active[2]. In the following we exemplify such a c-invariant in the tyreworld.

Example 3.1 (The Tyreworld)
The tyreworld domain [58] is an abstraction of the world we live in, excluding all situations that are not related to the exchange of a flat tire. Facts and actions of this world have intuitive names. For example, in(boot, nuts) and fastened(hub) are facts which state that the nuts of a rim are placed in the boot and that a hub is occupied by a wheel, respectively. Likewise, the naming of the tyreworld action tighten(nuts, hub, wrench) suggests that it is used to tighten the nuts on a hub, i. e., to delete the fact loose(nuts, hub) and to add tight(nuts, hub).

In our world, we know that physical objects are never in two places simultaneously and we expect objects to remain existent over time. Although we can imagine situations which violate these assumptions, e. g., after destroying an object, they always hold in the tyreworld. Hence, exactly one fact of the set $C_{nuts} = \{in(boot, nuts), have(nuts), loose(nuts, hub), tight(nuts, hub)\}$ *is true in every state of the tyreworld.* ■

By examining c-invariants and their associated actions we can identify similarities between all solutions to a given planning problem. Consider the tyreworld c-invariant C_{nuts} from the preceding example and a tyreworld problem where in the initial state the nuts are stored in the boot and in the goal they are tightly on the hub. Any solution to this problem first fetches the nuts out of the boot, puts them on the hub and tightens them, which means that any solution includes the actions $fetch(nuts, boot)$, $do\text{-}up(nuts, hub, wrench)$,

[1] In a world without the destruction of objects, such a state invariant corresponds to a set of facts of which *exactly* one is true.

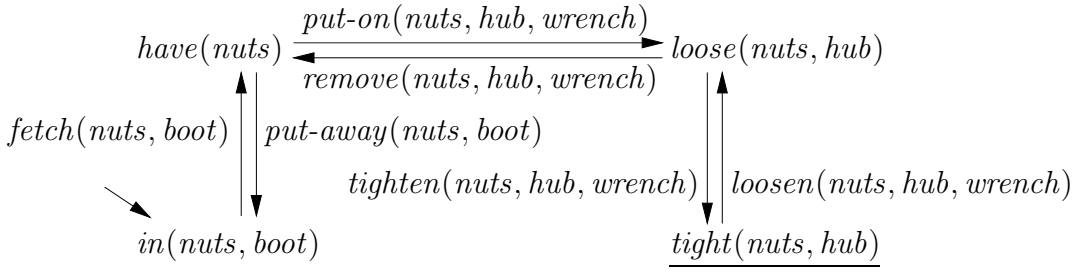
[2] Recall that we use the terms *true fact* and *active fact* synonymously. We prefer the term *true fact* to talk about facts of a state and the term *active fact* to talk about facts of a c-invariant independently of a specific state.

and $tighten(nuts, hub, wrench)$ in the given order. In other words, if $in(boot, nuts)$ is true in the initial state of the planning problem and $tight(nuts, hub)$ has to be true in the goal then this action sequence is present in the every solution to that problem. Furthermore, in each state during the execution of such a solution exactly one fact out of C_{nuts} is active, beginning with $in(boot, nuts)$ and ending with $tight(nuts, hub)$, and the changes of the active fact coincide with the action of the sequence.

These sequences of active facts of c-invariants during the execution of a plan suggests an analogy of c-invariants with labeled transition systems.

Example 3.2 (Analogy of c-Invariants with Transition Systems)
We can draw an analogy between c-invariants of an planning problem on the one hand and labeled transition systems on the other. A transition system is a four-tuple $\big(V, E, I, G\big)$, where V is a set of states and E is a set of labeled transitions, i. e., three-tuples $\big(v, v', l\big)$, where l is a label and $v, v' \in V$ are states; we say that transition l connects *state v to state v'. Furthermore, I and G are subsets of V that correspond to the initial states and the goal states of the transition system. Note that we use the term* state *in two different meanings: plan state and state of a transition system.*

Let us consider the tyreworld c-invariant $C_{nuts} = \big\{in(boot, nuts), have(nuts), loose(nuts, hub), tight(nuts, hub)\big\}$ (cf. Example 3.1 on the preceding page), where the nuts are initially in the boot and are to be put tightly on the hub. The states of the corresponding transition system are the facts of C_{nuts} and its transitions are labeled with the actions of the tyreworld that change the active fact of C_{nuts}. The sets I and G of this transition system comprise the respective fact of C_{nuts} that is true in the initial state and the goal of the given problem. States in I and G are depicted by an arrow without outgoing node and by being underlined, respectively.



If fact $in(boot, nuts)$ is true in the initial state of the planning problem and fact $tight(nuts, hub)$ is true in the goal then every solution to that problem corresponds to a path in the given transition system that begins with state $in(boot, nuts)$, that ends with state $tight(nuts, hub)$, and that uses the transitions whose labels are actions of the solution. Likewise, for every path in the transition system that connects $in(boot, nuts)$ to $tight(nuts, hub)$ there is a solution to the planning problem which has the action sequence $\langle fetch(nuts, boot), put\text{-}on(nuts, hub, wrench), tighten(nuts, hub, wrench)\rangle$ as subsequence, which is given by the sequence of transition labels. ■

This analogy motivates us to use some terminology of transition systems for c-invariants which is formally introduced in detail in the subsequent sections. In parlance of transition

systems, a path is a sequence of nodes that changes the state of a transition system to another via the execution of transitions. For c-invariants we use a related notion of path: A *path* through a c-invariant is a sequence of actions that changes the active fact of this c-invariant from the fact at its beginning to the fact at its end; we say that the path *connects* the first to the second fact and we call an action a *path-action* if it can be part of a path. Note that we define paths via actions instead via facts because we want to reference action sequences unequivocally. Fact sequences do not suffice for this purpose: As the graph of a transition system can be non-simple, a sequence of facts can correspond to several sequences of labeled transitions, i. e., paths.

The analogy between c-invariants and transition systems suggests a divide and conquer approach to solving planning problems: Obviously, it is easy to find a path through a c-invariant that connects two given facts. If, in addition, all facts of a planning domain are members of c-invariants then a solution to any problem of that domain is a combination of paths that connect the active facts of these c-invariants in the initial state to their active facts in the goal. The respective divide and conquer solution strategy has two steps: First, find a path through each of these c-invariants that comprises all path-actions of this c-invariant that are present in a solution, i. e., a path that connects the initial state with the goal. Then, combine these paths to a solution in a conflict free way. If both steps are computationally easy then this approach results in a competitive planning technique.

Unfortunately, this divide and conquer planning strategy based on c-invariants and paths fails because of two reasons: Firstly, the problem of finding a single path that is part of a solution to a planning problem is as hard as planning itself; see Section 3.1.8 on Page 34 for details. Secondly, the paths in a plan are not independent of each other: Some actions correspond to transitions in several c-invariants and their execution changes the state of these c-invariants simultaneously. Consequently, the merging of two paths requires that they use the same multiple transitions in the same order, thus preventing an easy division of planning problems by c-invariants into simpler, independent problems.

Because of the interdependency of paths in plans, the use of paths for planning requires to examine these interactions in depth. An interaction between two paths is the result of an action that is path-action of two c-invariants simultaneously. If such an action is in a plan then it is present in the paths through these two c-invariants; in other words, the paths have a common action. For a single path, such common actions appear as side-effect of that path: The execution of the common action requires the second c-invariant to be in a certain state and afterwards this c-invariant is in another certain state. These side-effects of paths are similar to paths themselves, e. g., they can be described as sequences of actions: If two adjacent actions in a path affect the same c-invariant and the transition of the first ends with the same fact that the second begins with then the combined transition, i. e., the side-effect on this c-invariant, connects the precondition of the first to the postcondition of the second. The side-effect analogy motivates our use of the term *sidepath* for such subsequences of paths and *sidepath-action* for the actions of a sidepath. Let us exemplify sidepaths and sidepath-actions in the tyreworld.

Example 3.3 (Side-Effects of Path-Actions and Paths)
The following tables give some actions and facts of the tyreworld. The abbreviations used for the actions denote their effects on a tyreworld c-invariant. For example, the action $o_{32} = undo(nuts, hub, wrench)$ *consumes the fact* $f_3 = loose(nuts, hub)$ *and adds fact* $f_2 = have(nuts)$ *of the c-invariant that*

corresponds to the location of the nuts. Further c-invariants of the tyreworld are $C_{cranked} = \{f_6, f_7\}$*, which indicates whether the hub is cranked up or not,* $C_{wrench} = \{f_8, f_9\}$*, which comprises the locations of the wrench, and* $C_{hub} = \{f_{10}, f_{11}\}$*, which indicates whether something is on the hub or not.*

	fact		*fact*		*fact*
f_1	*in(boot, nuts)*	f_5	*open(boot)*	f_9	*in(boot, wrench)*
f_2	*have(nuts)*	f_6	*not-on-ground(hub)*	f_{10}	*unfastened(hub)*
f_3	*loose(nuts, hub)*	f_7	*on-ground(hub)*	f_{11}	*fastened(hub)*
f_4	*tight(nuts, hub)*	f_8	*have(wrench)*	f_{12}	*have(jack)*

	action	*preconditions*	*added facts*	*deleted facts*
o_{12}	*fetch(nuts, boot)*	f_1, f_5	f_2	f_1
o_{21}	*put-away(nuts, boot)*	f_2, f_5	f_1	f_2
o_{23}	*do-up(nuts, hub, wrench)*	f_2, f_6, f_8, f_{10}	f_3, f_{11}	f_2, f_{10}
o_{32}	*undo(nuts, hub, wrench)*	f_3, f_6, f_8, f_{11}	f_2, f_{10}	f_3, f_{11}
o_{34}	*tighten(nuts, hub, wrench)*	f_3, f_7, f_8	f_4	f_3
o_{43}	*loosen(nuts, hub, wrench)*	f_4, f_7, f_8	f_3	f_4
o_{67}	*jack-up(hub)*	f_6, f_{12}	f_7	f_6, f_{12}

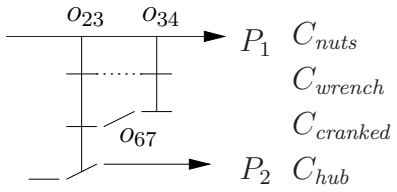
The action do-up(nuts, hub, wrench) places some nuts on a hub. This action also yields a fastened hub by consuming fact unfastened(hub) and adding fact fastened(hub) of the c-invariant C_{hub}*. Thus, making a hub fastened is a side-effect of placing nuts on the hub.*

Example 3.2 on Page 17 introduced the tyreworld path $\langle$*fetch(nuts, boot), do-up(nuts, hub, wrench), tighten(nuts, hub, wrench)*$\rangle$ *through c-invariant* C_{nuts} *that puts the nuts on a hub and tightens them. Its actions depend on and affect the c-invariants* C_{hub}*,* C_{wrench}*, and* $C_{cranked}$*, i. e., the fastening of the hub, the location of the wrench, and whether the hub is cranked or not. In other words, the actions of the path have side-effects in the corresponding c-invariants. For a given c-invariant the combination of these side-effects is similar to a path through that c-invariant. For example, both actions do-up(nuts, hub, wrench) and tighten(nuts, hub, wrench) depend on the same fact of* C_{wrench} *and so does the path that uses them. We call such combined side-effects of a path* sidepath *and the actions that comprise a sidepath we call* sidepath-action. ■

If we examine whether a plan is a solution to a planning problem we are not interested in any of its actions in particular; we rather want to know whether the plan solves the problem. Likewise, our interest does not lie in the intermediate states during the execution of a plan besides their occurrence during the execution of a solution. The same is true for paths: The actual action sequence of paths and sidepaths is of minor interest, as is their sequence of active facts. The important aspect is whether several of them can be combined in a conflict-free way, i. e., their interdependence and interaction. We now introduce a graphical representation that allows to depict these important aspects of paths and allows to abstract from the unimportant ones.

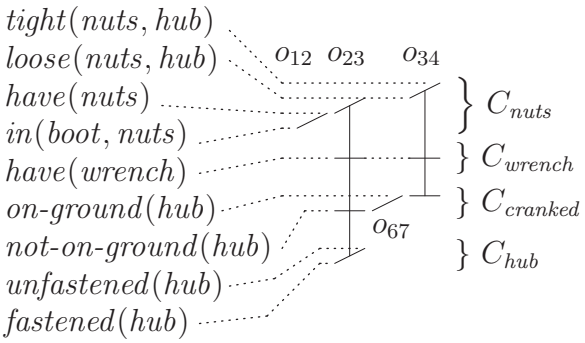
Example 3.4 (Abstract Representation of Paths and Sidepaths)
Throughout this work we use graphs to depict paths, sidepaths, and their interdependencies. Consider the plan $\langle$*fetch(nuts, boot), do-up(nuts, hub, wrench),*

jack-up(hub), *tighten*$(nuts, hub, wrench)\big\rangle = \big\langle o_{12}, o_{23}, o_{67}, o_{34}\big\rangle$ *of the tyreworld. It has the path* P_1 *through c-invariant* C_{nuts} *as subsequence, which has the action sequence* $\big\langle$*do-up*$(nuts, hub, wrench)$,*tighten*$(nuts, hub, wrench)\big\rangle = \big\langle o_{23}, o_{34}\big\rangle$. *We know from Examples 3.2 on Page 17 and 3.3 on Page 18 that this path depends on and interacts with several other c-invariants, i. e.,* C_{hub}, C_{wrench}, *and* $C_{cranked}$. *The following figure gives a simple graphical representation of these interactions and dependencies:*



In such a graph, the actions of the plan are arranged from left to right. Facts are located horizontally on different heights, with facts of a c-invariant grouped to a strip (note that the following graph depicts the same situation and shows these facts and strips explicitly). A path (or a sidepath) is depicted in the strip corresponding to its c-invariant. They are given as horizontal arrow or line, respectively, if the facts that it passes through are not relevant. Otherwise, we use short diagonal lines for path-actions and short horizontal lines for sidepaths-actions. These lines start and end at the precondition and postcondition of the action, respectively. A dotted line denotes that several actions have the same pre- and postcondition. With a vertical line we point out that an action is path- and sidepath-action of several c-invariants simultaneously. The small gaps in sidepaths depict boundaries of subsequences that are of interest.

To make it clearer how to read such graphs, the following graph depics the same situation as the previous one and shows the c-invariants with their respective strips and facts.



This graph immediately reveals that the path through C_{nuts} *with action sequence* $\big\langle$*do-up*$(nuts, hub, wrench)$, *tighten*$(nuts, hub, wrench)\big\rangle$ *has a sidepath through* C_{wrench} *of length two, a sidepath through* $C_{cranked}$ *of length one, and two sidepaths through* C_{hub}*, each of length one.* ■

The aim of this work is to present techniques that use paths for planning. The first technique finds paths that have the same interaction, i. e., decides for a path whether there is another path such that whenever the first path is part of a solution then there is another solution to the same problem that differs from the first solution solely by using the second path instead of the first. Simply said, it finds paths which can replace other paths in solutions. This knowledge has the potential of reducing the search space of a planning system by decreasing the number of actions, action sequences, paths, and solutions under consideration. Furthermore, if we prefer the replacement of longer paths by shorter ones then it increases the chance of finding short solutions. We call this technique *path replacement* and present it in Chapter 4 starting on Page 39.

Building on path replacement, Chapter 5 beginning on Page 63 introduces *path reduction*, a constructive preprocessing technique for planning problems. For a given planning problem, path reduction constructs a set of paths such that the actions of these paths can form a shortest solution to the problem; actions that are not used by paths in the set are irrelevant and can be removed from the problem. The rest of this section provides a detailed and formal analysis of c-invariants and paths in planning.

3.1.2 c-Invariants – A Class of State Invariants

The class of c-invariants is a sub-class of state invariants and is characterized by a set of facts of which at most one is true in a state. In deviation from the standard representation of state invariants as formulas on states, we identify a c-invariant with its set. This representation is convenient as states itself are sets of facts and we can use set notation to determine the fact of a c-invariant that is true in a state. Furthermore, the set representation facilitates the analogy between c-invariants on the one hand and transition systems on the other, as presented in Example 3.2 on Page 17.

Usually, a state invariant, e. g., in the situation calculus, is a conjunction of disjunctions $\big(\neg\,\mathrm{holds}(f_1, s) \vee \neg\,\mathrm{holds}(f_2, s)\big)$ for every pair f_1, f_2 of different facts of the invariant. Hereby, the predicate $\mathrm{holds}(f, s)$ is read as fact f is true in state s. It is apparent that there is no easy connection between such formulas and the view of c-invariants as transition systems. Furthermore, the size of such a formula grows quadratic in the size of the described fact set. In contrast, the set representation grows linear in the size of the c-invariant and resembles directly the node set of a graph.

The following definition of c-invariants as fact set shows their relation to properties of the actions of a planning domain. Remember that a global sink of a directed graph is a vertex that is reachable from all other vertices of the graph.

Definition 3.1 (c-Invariants)
A c-invariant C *of a planning problem* $\Psi = \big(\mathcal{O}, \mathcal{I}, \mathcal{G}\big)$ *is a subset of* $\mathcal{F}_\Psi$ *that has each of the following properties:*

1. *Set* C *is of size two or more.*
2. *Every action* $o \in \mathcal{O}$ *adds either zero or one fact of set* C*.*
3. *Every action* $o \in \mathcal{O}$ *that adds a fact of set* C *also consumes at least one fact of* C*.*

4. *The directed graph* $\big(C, E\big)$ *has a global sink, where the nodes of the graph are the facts in* C *and* E *is the set* $\big\{(f, f') \;\big|\; o \in \mathcal{O},\ cons(o) \cap C = \{f\},\ add(o) \cap C = \{f'\}\big\}$.

From now on, we denote with $\mathcal{C}$ *a set of c-invariants. The fact set* $\mathcal{F}^{\mathcal{C}}$ *denotes the union of all c-invariants in* $\mathcal{C}$. *With* $\mathcal{C}_\Psi$ *we denote the set of all c-invariants of planning problem* Ψ. *The fact set* $\mathcal{F}^{\mathcal{C}}_\Psi$ *is the union of all c-invariants of* Ψ, *and* $\mathcal{F}^{\overline{\mathcal{C}}}_\Psi = \mathcal{F}_\Psi \setminus \mathcal{F}^{\mathcal{C}}_\Psi$ *is the set of facts of the planning problem that do not belong to any c-invariant of* Ψ. ◇

A usual plan state includes a fact of every c-invariant C of its planning domain: The initial state specifies the fact of C that is true in the state before the first action and the goal specifies the one after the last. Actions of a plan either change the fact of C in states or do not affect C. This leads to the view of c-invariants as transition system, where the fact of C in a plan state is the state of the transition system and the actions that consume and add facts of C in plan states are its transitions (cf. Example 3.2 on Page 17). To avoid confusion between the two uses of the word "state", plan state and state of a transition system, we introduce a new terminology for the latter.

Definition 3.2 (Active Fact of a c-Invariant in a State)
We say that fact f *is the* active fact *of a c-invariant* C *in a plan state* s *if* f *is in* s, *i. e.,* $f \in C \cap s$. *Hereby* C *is a c-invariant of a planning problem* Ψ *and* s *is a state of* Ψ. *We say that* f *is the active fact of* C after *an action* o *if* $post(o) \cap C = \{f\}$. *We also say that* f *is the active fact of* C before *action* o *if* $pre(o) \cap C = \{f\}$. ◇

In other words, the active fact of a c-invariant C in a plan state denotes the fact of C that is true in that state. The notion of an active fact before and after an action allows to conveniently talk about the states before and after the execution of an action. Note that we only refer to the state of a c-invariant before an action o if this fact is unique, i. e., if $|pre(o) \cap C| = 1$.

Usual plan states have one active fact for each c-invariant of the planning domain. Nevertheless, there can be degenerated states with zero active facts, as well as with two or more active facts of a c-invariant. We distinguish these two kinds of abnormal states because of their different consequence on the future of a c-invariant C: In the first case, no fact of C will ever become true again. One could say that such facts represent a dead part of a state, or that the state *destroys* the c-invariant C. In the second case, the facts of C can still change their truth value and it is possible to reach a state which has exactly one active fact of c-invariant C.

Definition 3.3 (Observation, Violation, and Destruction of a c-Invariant)
A c-invariant C *is* destroyed *in a state* s *if* $|C \cap s| = 0$ *and it is* violated *in* s *if* $|C \cap s| \geq 2$. *Otherwise,* s observes C. ◇

In the following, we will elaborate on c-invariant destruction but not on their violation: An observed c-invariant can become destroyed in a future state but can never become violated. Because we can eliminate c-invariants of a planning domain that are violated by the initial state of a planning problem, we will never encounter a violated c-invariant. This is different with c-invariant destruction, which can be inevitable in order to solve a planning problem. Hence, we will closely examine the properties of c-invariant destruction.

With the notion of destruction and violation of c-invariants we are now ready to explain the motivation behind Points 1 and 4 of the definition of c-invariants: They exclude

degenerated fact sets from being a c-invariant: The first point excludes fact sets of size one, which would satisfy the definition trivially. Point 4 prevents unnecessarily violated and destroyed c-invariants by requiring the graph of a c-invariant to have a global sink, i. e., a fact that is reachable from all other facts. If c-invariants were defined without this restriction then any union of two disjoint c-invariants would be a c-invariant, too. Let us give an example for the consequences of such a definition:

Example 3.5 (Degenerated c-Invariants)
Point 4 of the definition of c-invariants excludes degenerated fact sets from being a c-invariant. Let us consider Definition 3.1 on Page 21 without the restriction of Point 4 and let C_1 and C_2 be two c-invariants according to that definition.

As a consequence, if C_1 and C_2 are disjoint then their union $C_{1,2} = C_1 \cup C_2$ is a c-invariant, too, which can be easily verified by checking against Points 1 to 3. Unfortunately, this c-invariant has unwanted properties: Any state observing both c-invariants C_1 and C_2 violates $C_{1,2}$. Furthermore, if a state observes the c-invariant $C_{1,2}$, i. e., only one fact of $C_{1,2}$ is true then exactly one of c-invariants C_1 or C_2 is observed by that state and the other is destroyed.

The postulation of a global sink prevents many unions of c-invariants, including $C_{1,2}$, to be c-invariants themselves, so that all c-invariants of a domain can be in a valid state at the same time. In fact, if the union of two c-invariants yields another c-invariant then one of them must be a subset of the other. ■

Why do we restrict ourselves to c-invariants as opposed to other classes of state invariants? Many of such classes correspond to properties of domains other than fact consumption and hence are not amenable to the same exploitation as is the class of c-invariants. As an illustration, the following example presents a class of state invariants for which the view as transition system is inappropriate.

Example 3.6 (ad-Invariants)
ad-Invariants are a class of state invariants, similar to c-invariants. A set of facts D is an ad-invariant if every action that adds a fact of set D deletes all others. The pairing between an added ("a") and deleted ("d") facts in actions motivates their name ad-invariant. Clearly, if at most one fact of set D is in a state s then this holds for all reachable states. An example is the fact set $D = \{f_1, f_2, f_3, f_4\}$ which is an ad-invariant of the action set $\big\{\big(\emptyset, \{f\}, D \setminus \{f\}\big) \bigm| f \in D\big\}$.

In contrast to c-invariants, the active fact of an ad-invariant does not depend on any of its previous active facts and its graph is fully connected. Therefore, the changes to the active fact of an ad-invariant are independent of the ad-invariant itself and the view of ad-invariants as transition system is not appropriate. ■

In the following, we examine closely the structural features of planning domains that are related to the existence of c-invariants.

3.1.3 Path-Actions and Paths

c-Invariants are of interest beyond being a class of state invariants. When viewing them as transition system, the execution of a plan changes the active fact of a c-invariant C in the initial state of a problem to the one in the goal state. Hereby, the sequence of actions that add a fact of C corresponds to a path in the transition system. Hence, we can see the transition system of a c-invariant as relaxation of the overall planning problem and the path through this transition system as relaxation of the solution.

Reasoning about such paths instead of single actions is advantageous because they allow to abstract from their intermediate actions and active facts, as it is often sufficient to know their start and their stop fact. In short, paths are larger building blocks of plans than single actions and are less specific than partial sequences of solution plans. In the following, we give the basic definitions and terminology for reasoning about paths.

Definition 3.4 (Path-Actions and Paths)
A path-action *of a c-invariant C is an action o that adds a fact of C, i. e., $add(o) \cap C \neq \emptyset$. If $\Psi = \big(\mathcal{O}, \mathcal{I}, \mathcal{G}\big)$ is a planning problem then $\mathcal{O}^{\mathcal{C}}_{\Psi}$ is the set of all path-actions of Ψ and $\mathcal{O}^{\overline{\mathcal{C}}}_{\Psi}$ is the set $\mathcal{O} \setminus \mathcal{O}^{\mathcal{C}}_{\Psi}$.*

A path *P is a pair $\big(T, C\big)$ of a sequence T of actions and a c-invariant C that has the following properties:*

1. *All actions in sequence T are path-actions of c-invariant C.*
2. *For all adjacent actions o, o' in sequence T it holds that $post(o) \cap C = pre(o') \cap C$.*

We call P proper *if T is not the empty sequence. Otherwise, we call P the* empty *path.* ◇

Let us ease the upcoming analysis of the properties of paths by defining some terminology and abbreviations. In particular, if we talk about a path-action o in path $P = \big(T, C\big)$ without specifying the c-invariant of which o is path-action, then o shall be a path-action of c-invariant C. Furthermore, we use the terminology for sequences to talk about paths. For example, if we use path P in a phrase that has been defined for sequences but not for paths then we read it as the sequence T. The following terminology allows us to refer to the beginning and the ending, and other properties of paths.

Definition 3.5 (Beginning and Ending of a Path)
Let $P = \big(T, C\big)$ be a path. We say that P is a path through c-invariant C and that P begins *with fact f, where $pre(T) \cap C = \{f\}$. We say that path P* ends *with fact f' and* connects *fact f to fact f' if $post(T) \cap C = \{f'\}$. We say that path P is* circular *in c-invariant C if facts f and f' are the same. If we just say that path P is circular, i. e., omit the c-invariant it is circular in then we always refer to the c-invariant that P goes through. We also say that path P* passes through *a fact f if f is a fact in c-invariant C and either P begins with fact f or there is an action o in P with $f \in post(o)$.* ◇

In the following, we first examine path and their properties in isolation. Nevertheless, if we want to denote that a path is part of a plan then we say that it is *in* that plan.

Definition 3.6 (Path in a Plan)
We say that path (T, C) *is* a path in *plan* T' *if* T *is a partial sequence of* T' *and all path-actions of c-invariant* C *in* T' *between the beginning and the ending of sequence* T *are actions in* T. ◇

Note that we have two ways of referring to the relation between an action and a path in the same plan: Firstly, we can say that an action is *in* a path and secondly that an action lies *during* a path: In the first case, the action is both part of the path and of the plan. In the second case, we read path as the action sequence of the path. Then Definition 2.6 on Page 8 states that the action is in the plan, too, but it might or might not be part of the path. Now let us give an example of paths and the use of the corresponding terminology in the tyreworld.

Example 3.7 (The Tyreworld, continued)
Action $fetch(nuts, boot)$ *adds the fact* $f_2 = have(nuts)$*, so it is a path-action of the c-invariant* C_{nuts} *that corresponds to the status of the nuts. In contrast, action* $o_{98} = fetch(wrench, boot) = \big(\{f_5, f_9\}, \{f_8\}, \{f_9\}\big)$ *is not a path-action of c-invariant* C_{nuts}*. As before, the index of action* o_{89} *indicates that it connects fact* f_8 *to fact* f_9*.*

The tuple $P = \big(\langle o_{23}, o_{34}\rangle, C_{nuts}\big)$ *is a path through c-invariant* C_{nuts} *that connects fact* $have(nuts)$ *to fact* $tight(nuts)$ *and additionally passes through fact* $loose(nuts, hub)$*. It does not pass through* $in(nuts, boot)$*. If* $\big(\mathcal{O}, \{f_2, f_9\}, \{f_4, f_8\}\big)$ *is a planning problem in the tyreworld then* $\langle o_{23}, o_{98}, o_{34}\rangle$ *is a solution of this problem and* P *is a path in that solution.* ■

Unfortunately, there is more to paths than the facts they connect. If not, they would be amenable to a simple divide and conquer strategy. In most cases, such an approach is prevented by an interaction between paths, of which the most important we term sidepath.

3.1.4 Sidepaths, the Side-Effects of Paths

An action can change the active fact of two or more c-invariants simultaneously. In other words, it can be path-action of more than one c-invariant. If such an action is in a plan then the plan has a path for every c-invariant of which the action is a path-action. For planning problems Ψ with $\mathcal{F}_\Psi^{\overline{\mathcal{C}}} = \emptyset$, i. e., all facts of the problem are in a c-invariant, we can say that a plan is a combination of a collection of paths. If an action in such a plan is path-action of two c-invariants then the corresponding paths in the collection have a common action. For a single path, such common actions appear as side-effect of that path (cf. Example 3.3 on Page 18).

The side-effects of paths are similar to paths themselves: If two adjacent actions in a path affect the same c-invariant C and the corresponding postcondition of the first is a consumed effect of the second then their combined side-effect on c-invariant C connects the precondition of the first to the postcondition of the second. The analogy to side-effects motivates our use of the term *sidepath* for such subsequences of paths. As we are primarily interested in the active facts of c-invariant C before and after the execution of the path, we require sidepaths to be as long as possible.

Definition 3.7 (Sidepath-Actions and Sidepaths)
Let o be an action and C be a c-invariant. Action o is a sidepath-action *of C if o has a fact of C among its preconditions, i. e., $pre(o) \cap C \neq \emptyset$. We call o a* proper *sidepath-action of C if o is a sidepath-action but not a path-action of C.*

A sidepath *Q of a path $P = (T, C)$ is a pair (T', C') of a non-empty, partial sequence T' of sequence T and a c-invariant C' (not necessarily different from C), such that*

1. *All actions in sequence T' are sidepath-actions of c-invariant C'.*
2. *For all adjacent actions o, o' in sequence T' it holds that $post(o) \cap C' = pre(o') \cap C'$.*
3. *For all adjacent actions o, o' in sequence T' it holds that they are not separated by a sidepath-action of c-invariant C' in sequence T. Furthermore, they are not separated by an action o'' in T with $post(o) \cap C' \subseteq del(o'') \cap C'$.*
4. *There is no partial sequence T'' of sequence T, such that T' is a proper subsequence of T'' and the previous three points are true for T''.*

A proper *sidepath Q of a path P is a sidepath of P and $Q \neq P$.* ◇

Points 1 and 2 of the previous definition are similar to the definition of paths. Point 3 prevents the interruption of sidepaths: The first part prevents intermediate sidepath-actions of C' that are not in the sidepath. The second part prevents intermediate actions o'' that delete a fact C'; we say that such actions *endanger* C'. Section 3.1.6 on Page 29 introduces c-invariant endangerment and motivates the special treatment of endangering actions in the definition of sidepaths. Point 4 prevents proper subsequences of sidepaths from being sidepaths themselves. In other words, sidepaths are always as long as possible.

The definition of sidepaths states that a sidepath-action of a c-invariant C is not required to be a path-action of C; it only needs a fact of C as precondition. Such sidepath-actions result in a side-effect of their path, too: Before the execution of the path, the c-invariant of the sidepath is required to have certain fact active, and afterwards the active fact of this c-invariant is known. In other words, a sidepath-action determines the active fact of its c-invariant in the same way as a path-action does.

Every path-action of a c-invariant C is also a sidepath-action of C. This follows from the definition of c-invariants, which states that $add(o) \cap C \neq \emptyset$ implies $pre(o) \cap C \neq \emptyset$ for every action o. As a result, every path is its own sidepath. To underline the similarity between paths and sidepaths, we use the same terminology with sidepaths as with paths, e. g., to address their c-invariants and their actions. Let us give an example of the sidepaths of a path:

Example 3.8 (The Tyreworld, continued)
Reconsider path $P = (\langle o_{23}, o_{34} \rangle, C_{nuts})$ as given in Example 3.7 on the preceding page. Its execution places the nuts on the hub and tightens them. Path P has proper sidepaths through the following c-invariants: $C_{cranked} = \{f_6, f_7\}$, which indicates whether the hub is cranked up or not, $C_{wrench} = \{f_8, f_9\}$, which comprises the locations of the wrench, and $C_{hub} = \{f_{10}, f_{11}\}$, which indicates whether something is on the hub or not. In detail, the proper sidepaths of P are $(\langle o_{23} \rangle, C_{cranked})$, $(\langle o_{34} \rangle, C_{cranked})$, $(\langle o_{23}, o_{34} \rangle, C_{wrench})$, and $(\langle o_{23} \rangle, C_{hub})$. In addition, P is its own sidepath, though not a proper one. ■

The succession of path- and sidepath-actions in sidepaths is important for analyzing paths. In our view of plans as collection of paths, actions that are path-actions of two or more c-invariants are common actions of their respective paths. In an attempt to combine the paths of a collection to a conflict-free plan, such actions pose a restriction on the placement of their paths: Obviously, an action that is path-action of several c-invariants has to be placed to the same time step for all paths it is in.

Sidepath-actions pose a similar restriction on the combination of paths in a collection, as sidepath-action can only be placed in a time step where its required facts are active. This can be seen with the path given in Example 3.7 on Page 25: Action o_{23} puts the nuts onto a hub, which requires to have a wrench. Action o_{89} places the wrench into the boot, so action o_{23} has to be executed before action o_{89}. In a sidepath, the position of the first and last path-action is of particular interest. The following definition gives a corresponding terminology that allows to address those actions. Example 3.4 demonstrates this terminology and introduces graphs that depict configurations of paths and sidepaths.

Definition 3.8 (The Core and Fringes of a Sidepath and Its Induced Path)
The induced path *of a sidepath Q is the path $\big(T', C\big)$, where $Q = \big(T, C\big)$ and T' is the sequence of all path-actions of C in T. We call sidepath Q* coreless *if its induced path is empty; otherwise we say Q has a* core*.*

If Q has a core then let T_2 be the shortest subsequence of sidepath Q that contains sequence T' and let $T = T_1 \circ T_2 \circ T_3$. Then T_1 is the left fringe *of sidepath Q, T_2 is its* core*, and T_3 is the* right fringe *of Q. If Q is coreless then we say that all actions of Q are in its left fringe.* ◇

Fringes are successions of proper sidepath-actions, i. e., their execution does not change the active fact of the sidepath. The left and the right fringe of a sidepath are its prefix and postfix before and after its core, respectively. Let us demonstrate the use of this notation in the tyreworld domain.

Example 3.9 (The Tyreworld, continued)
Recall path $P = \big(\langle o_{23}, o_{34}\rangle, C_{nuts}\big)$ from Example 3.8 on the facing page. Its execution places the nuts on the hub and tightens them. Two of its sidepaths are $Q_1 = \big(\langle o_{23}\rangle, C_{hub}\big)$ and $Q_2 = \big(\langle o_{23}, o_{34}\rangle, C_{wrench}\big)$. The first sidepath fastens the hub. As action o_{23} is a path-action of C_{hub}, the left and the right fringe of Q_1 are empty, sequence $\langle o_{23}\rangle$ is the core of Q_1 and $\big(\langle o_{23}\rangle, C_{hub}\big)$ is the induced path of Q_1. Sidepath Q_2 states that we have to have the wrench during the execution of path P. As the status of the wrench does not change during the execution of P, sidepath Q_1 is coreless and its induced path is empty. ■

3.1.5 Overlapping c-Invariants

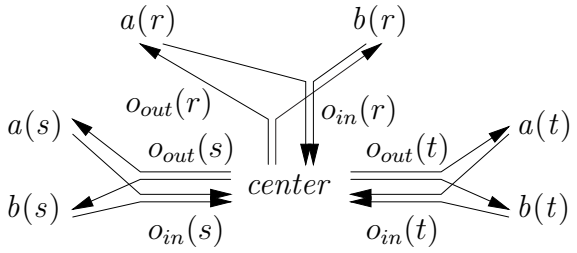
A special case of sidepaths results from c-invariants with non-empty intersection, we call such c-invariants *overlapping*: If two c-invariants C_1 and C_2 have a non-empty intersection $C_{1\cap 2}$ then any path $\big(T, C_1\big)$ that passes through facts in $C_{1\cap 2}$ has a sidepath $\big(T, C_2\big)$ and vice versa. These paths and sidepaths have the same action sequence and differ only in their c-invariant; we can view them as a single path that goes through C_1 and C_2 simultaneously. If an action o connects a fact in set $C_1 \setminus C_{1\cap 2}$ to a fact f in c-invariant C,

then o connects a fact in $C_2 \setminus C_{1\cap 2}$ to the same fact f, too. We say that action o *merges* paths through c-invariants C_1 and C_2.

Example 3.10 (The Tyreworld, continued)
The tyreworld c-invariants $C_{nuts} = \{f_1, f_2, f_3, f_4\}$ *and* $C_{hub} = \{f_3, f_4, f_{10}\}$ *comprise the locations of the nuts and the status of the hub, respectively. These c-invariants overlap because they have the common subset* $C = \{f_3, f_4\}$. *The action do-up(nuts, hub, wrench) places the nuts on the hub and occupies the hub, thus merging the paths through c-invariants* C_{nuts} *and* C_{hub}. ■

Overlapping c-invariants can yield domains with an exponential number of c-invariants compared to the size of the domain. The following example shows a domain with $2n$ merging actions and 2^n c-invariants.

Example 3.11 (The Explode Domain)
The explode domain consists of two classes of actions: Instantiations of action $o_{in}(?x)$ *consume a pair a(?x), b(?x) of facts and add the fact center, and instantiations of* $o_{out}(?x)$ *consume center and add a pair a(?x), b(?x) of facts. Hereby, the parameter ?x is substituted by elements of a finite set of objects. Therefore, an explode domain with n objects has n actions* $o_{in}(?x)$ *and n actions* $o_{out}(?x)$, *yielding a total domain size of* $2n$ *actions and* $2n+1$ *facts. The following graph denotes merging actions by arrows that are partially parallel.*



Let us examine the explode domain with three objects r, s, and t: Every action that adds fact with object r consumes fact center; likewise every action that consumes a fact with object r adds fact center. After verifying the remaining points of the Definition 3.1, we see that $\{a(r), a(s), a(t), center\}$ *is a c-invariant of this domain. In fact, every element of the cross product* $\{a(r), b(r)\} \times \{a(s), b(s)\} \times \{a(t), b(t)\} \times \{center\}$ *is a c-invariant, which results in eight c-invariants for the explode domain with three objects.* ■

Obviously, a technique that is based on c-invariants can show exponential behavior if it is applied to a planning domain whith has an exponential number of c-invariants compared to its size. Fortunately, the techniques that are presented in this work need only a subset of such c-invariants that covers the set $\mathcal{F}^{\mathcal{C}}$. At the end of this section we give a normalization of planning problems that eliminates the supernumerous c-invariants. Note that the choice of this subset is important: For an explode domain with size n, the smallest covering set consists of two c-invariants whereas the largest one, where all members have a unique element, has size n.

3.1.6 The Destruction of c-Invariants

By now, we have assumed that if the initial state of a planning problem observes a c-invariant of its domain then all reachable states observe it, too. This does not hold in general: An action o can delete a fact of a c-invariant C without adding another fact of C; in other words, action o can *destroy* c-invariant C. The destruction of a c-invariant in a plan has the consequence that no path- or sidepath-action of c-invariant C can be in that plan after action o. Furthermore, if the plan is a solution to a planning problem then the goal of this problem does not include a fact of c-invariant C.

c-Invariant destruction is uncommon in well known planning domains. For example, the blocks world and the tyreworld do not show this feature. Nevertheless, if a planning technique is aimed at general planning problems then it has to handle the destruction of c-invariants, as there are planning problems for which every solution destroys a c-invariant. Thus in the following, we examine c-invariant destruction for reasons of completeness.

We can distinguish two kinds of actions that destroy c-invariants: Firstly, an action can consume a fact of a c-invariant C without adding another. Here, the consumed fact has to be active in a state before the application of the destroying action, thus C is destroyed for sure. Such an action has the destroyed fact in its precondition, hence it is a sidepath-action of the c-invariant it destroys. Obviously, it is always the last action of a sidepath through its destroyed c-invariant and there is no later sidepath-action of C in its path or plan. The second kind of c-invariant destruction is that an action deletes a fact of a c-invariant C but does not have it in its precondition, in other words, it does not consume the fact it destroys. Such an action has the potential of destroying C but without knowing the planning problem it remains open whether its execution in a plan actually destroys it or not. Hence, we say that such an action *endanger* the c-invariant.

A path-action cannot endanger or destroy its own c-invariant but it can endanger and destroy others. As we have said earlier, after the destruction of a c-invariant C no action can be a path- or sidepath-action of C. Thus, if an action o in a path P destroys a c-invariant C then action o is the last action of a sidepath Q of path P through c-invariant C. We use the following terminology for sidepaths which destroy their c-invariant.

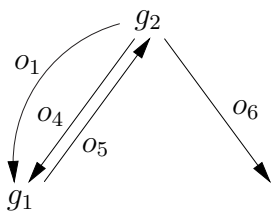
Definition 3.9 (Dead End of a Sidepath)
The dead end *of a sidepath is the last action of a sidepath that destroys its c-invariant. A sidepath* (T, C) *with dead end* o ends with the fact $pre(o) \cap C$. *If a sidepath does not destroy its c-invariant then it has a* regular ending. ◇

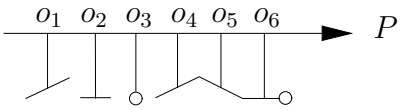
Note that according to this definition, the end of a sidepath with dead end refers to the last active fact that the sidepath passes through. This convention allows us to use the terminology of paths for sidepaths with dead end, e. g., that a sidepath with dead end is circular if its beginning and end are the same fact. Now, let us give an example of c-invariant endangerment and destruction.

Example 3.12 (Destruction and Endangerment of c-Invariants)
The domain $\{o_1, \ldots, o_6\}$, *as given in the following table, has two c-invariants* C_f *and* C_g. *The left graph shows c-invariant* C_g *as directed graph. It is not a simple graph because fact* g_1 *is connected to fact* g_2 *with two arcs, which represent the actions* o_3 *and* o_5.

In this domain we find an example of both kinds of c-invariant destruction: Action o_6 destroys the active fact g_2 of c-invariant C_g. The graph and the table depict this action by an arrow that leaves fact g_2 but does not enter another fact. Furthermore, action o_2 endangers c-invariant C_g. The graph does not depict action o_3, as the application of this action does not depend on the state of c-invariant C_g. The table shows the endangerment as bar over fact g_2.

c-invariants	o_1	o_2	o_3	o_4	o_5	o_6
$C_f = \{f_1, \ldots, f_7\}$	$f_1 \to f_2$	$f_2 \to f_3$	$f_3 \to f_4$	$f_4 \to f_5$	$f_5 \to f_6$	$f_6 \to f_7$
$C_g = \{g_1, g_2\}$	$g_2 \to g_1$	g_2	$\overline{g_2}$	$g_2 \to g_1$	$g_1 \to g_2$	$g_2 \to$





The right graph shows path $P = \big(\langle o_1, \ldots, o_6\rangle, C_f\big)$ together with its three sidepaths $\big(\langle o_1\rangle, C_g\big)$, $\big(\langle o_2\rangle, C_g\big)$, and $\big(\langle o_4, o_5, o_6\rangle, C_g\big)$. The first two sidepath have a regular ending. The third sidepath has a dead end, which we depict similar to a sidepath-action with an additional little circle at its end. Action o_3 endangers a fact of C_g but it is not in a sidepath of P; we depict it by a solitary little circle. This action endangers the fact with which the second sidepath ends and the third sidepath begins. In other words, action o_3 prevents the concatenation $\big(\langle o_2\rangle, C_g\big)$ and $\big(\langle o_4, o_5, o_6\rangle, C_g\big)$ from being a sidepath of path P. ■

We can extend the notion of c-invariant endangerment and destruction to sequences: A sequence destroys a c-invariant C if its execution destroys C for sure. Note that it is not necessary that the sequence contains an action which destroys C, it is sufficient that a sidepath-action o of C is followed by an action o' that endangers the fact $post(o) \cap C$. This case of c-invariant destruction is demonstrated by the sequence $\langle o_2, o_3\rangle$ of the domain given in the previous example, which destroys c-invariant C_g.

A sequence endangers a c-invariant C if it contains an action that endangers C and, in addition, no action of the sequence is a sidepath-action of C. If there is a sidepath-action of C in a sequence then it is always known whether this sequence destroys C or not. For example, if o is a sidepath-action of C and o' endangers a fact of C that differs from $post(o) \cap C$ then $\langle o, o'\rangle$ does not destroy C. Formally, we define the notion of c-invariant endangerment and destruction of sequences as follows:

Definition 3.10 (Endangerment and Destruction of Facts and c-Invariants) *Let Ψ be a planning problem and T a plan of Ψ. Then* destroy(T) *is the set of facts that plan T* destroys *and* endanger(T) *is the set of facts that T* endangers*, where*

1. $\text{destroy}(T) = \big\{f \;\big|\; C \in \mathcal{C}_\Psi,\ f \in pre(T) \cap C,\ post(T) \cap C = \emptyset\big\}$ *and*
2. $\text{endanger}(T) = \big\{f \;\big|\; C \in \mathcal{C}_\Psi,\ pre(T) \cap C = \emptyset,\ f \in del(T) \cap C\big\}$,

respectively. We say that T destroys *or* endangers *a c-invariant* C *if a fact of* C *is in* destroy(T) *or in* endanger(T)*, respectively. As usual, we allow single actions instead of sequences of length one and (side)paths instead of sequences. Furthermore,* $\mathcal{F}^{end}$ *denotes a set of endangered facts and* $\mathcal{F}^{end}_{\Psi}$ *is the set of all facts that are endangered by actions of* Ψ*.* ◇

Point 3 of Definition 3.7 on Page 26 prevents endangering actions during sidepaths if they endanger the fact that the sidepath currently passes through. If such a situation is present in a path P, i. e., there are adjacent sidepath-actions o, o' of C in P with $post(o) \cap C = pre(o') \cap C$ and between o and o' there is an action o'' in P that endangers the fact $post(o) \cap C$, then o and o' will always be separated by path-actions of C in a plan T: Path P can only be part of T if the destruction of C by o'' is prevented by path-actions of C in T, i. e., an action o_1 between o and o'' with $post(o_1) \cap C \neq del(o'') \cap C$ and an action o_2 between o and o'' with $post(o_2) \cap C = pre(o') \cap C$. Because of actions o_1 and o_2, the technique path reduction would separate actions o and o' anyway (cf. Section 5.5 on Page 70). To prevent this difficulty in the first place, we define sidepaths such that o and o' are part of two different sidepaths right from the start.

3.1.7 The Unstructured Part of Planning Problems

Up to now, we have discussed c-invariants as well as their related facts and actions. We can say that these facts and actions have an inherent structure that gives rise to the c-invariants of their domain, which allows us to regard planning domains as comprised by two partitions: The one that is *structured* in respect to c-invariants and the one that is *unstructured* in this respect. Then the *structured part* of a planning problem Ψ consists of the facts in $\mathcal{F}^{\mathcal{C}}_{\Psi}$ and the actions in $\mathcal{O}^{\mathcal{C}}_{\Psi}$; the *unstructured part* are the sets $\mathcal{F}^{\overline{\mathcal{C}}}_{\Psi}$ and $\mathcal{O}^{\overline{\mathcal{C}}}_{\Psi}$. In this section, we discuss the interaction between these two parts.

Facts of a planning problem can be unstructured in respect to c-invariants and there are planning problems that do not have c-invariants at all. As such problems are not amenable to path reduction, we disregard problems of the latter kind. Nevertheless, our thrive for general purpose planning techniques requires the coverage of problems that have an unstructured part. Let us give an example for such a problem.

Example 3.13 (Planning Problem and Its Unstructured Part)
Consider the planning problem $\Psi = \big(\{o_1, \ldots, o_6\}, \mathcal{I}, \mathcal{G}\big)$*, where the domain is as given in the following table, the initial state* $\mathcal{I}$ *is the set* $\{f_1, g_2\}$*, and the goal* $\mathcal{G}$ *is the set* $\{f_3, g_2, g_3\}$*. In contrast to previous planning problems and action sets, problem* Ψ *has non-empty sets* $\mathcal{F}^{\overline{\mathcal{C}}}_{\Psi} = \{g_1, g_2, g_3\}$ *and* $\mathcal{O}^{\overline{\mathcal{C}}}_{\Psi} = \{o_5, o_6\}$*. The table shows this unstructured part of problem* Ψ *by giving the corresponding preconditions and effects of actions in an explicit way.*

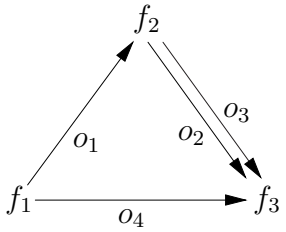
	o_1	o_2	o_3	o_4	o_5	o_6
$C_f = \{f_1, f_2, f_3\}$	$f_1 \to f_2$	$f_2 \to f_3$	$f_2 \to f_3$	$f_1 \to f_3$		
facts in $\mathcal{F}^{\overline{\mathcal{C}}}_{\Psi}$	*pre:* g_1		*add:* g_2		*add:* g_1 *del:* g_2	*pre:* f_2 *add:* g_3

■

The structured and the unstructured part of a planning domain interact during the execution of a plan. As we are interested in the analysis of the structured part of planning domains, we want to express the interaction between the two parts of a planning problem in terms of the structured part. In other words, we want to know path-actions and facts in $\mathcal{F}^{\mathcal{C}}$ on which the execution of a path-action can depend via the interaction with the unstructured part of the planning domain.

We can identify three different kinds of such interaction: (1) The structured part depends on the unstructured one if a path-action has a fact of $\mathcal{F}^{\overline{\mathcal{C}}}$ among its precondition: The application of this action depends on the prior application of actions in $\mathcal{O}^{\overline{\mathcal{C}}}$. The other two kinds of interaction result from dependencies of the unstructured on the structured part: Suppose o is an action in $\mathcal{O}^{\overline{\mathcal{C}}}$ with precondition f. (2) If f is a fact of c-invariant C then a path-action of C has to make fact f active before the application of action o. (3) If otherwise f is a fact in $\mathcal{F}^{\overline{\mathcal{C}}}$ and is added by a path-action o' of c-invariant C then the execution of action o' can be necessary for the applicability of action o. Let us exemplify these kinds of interactions.

Example 3.14 (Planning Problem and Its Unstructured Part, cont.) *Reconsider the planning problem Ψ of Example 3.13 on the page before. It has a single c-invariant $C_f = \{f_1, f_2, f_3\}$, which is depicted by the following graph:*



In order to find a solution to problem Ψ we have to respect the interaction between its structured and its unstructured part. Although action o_4 connects the active fact of c-invariant C_f in the initial state to its active fact in the goal, the sequence $\langle o_4 \rangle$ is not a solution to Ψ: Fact g_3 is false after its application. This fact is added by action o_6, which can only be applied if fact f_2 of c-invariant C_2 is active. In the previous paragraph, we referred to this as the second kind of dependency. Action o_1 cannot immediately reach this state because it depends on fact g_1. Action o_5 provides this fact but deletes goal fact g_2. Therefore, we cannot choose between the parallel actions o_2 and o_3 because only the latter adds fact g_2. The previous paragraph presented these dependencies as the first and the third kind, respectively. In the end, these interactions lead to the sole solution $\langle o_5, o_1, o_6, o_3 \rangle$ of problem Ψ. ■

If a fact $f \in \mathcal{F}^{\overline{\mathcal{C}}}$ is important for a path-action o, we want to know which facts of c-invariants and the execution of which path-actions could be necessary for reaching a plan state where fact f is true. In addition, we need to know all actions in the set $\mathcal{O}^{\overline{\mathcal{C}}}$ that could be necessary, too. We find these actions by backchaining over their added facts and preconditions, beginning with fact f. For example, we have to consider all actions in $\mathcal{O}^{\overline{\mathcal{C}}}$ that add fact f, then all actions in $\mathcal{O}^{\overline{\mathcal{C}}}$ that add preconditions of the former, a. s. o. The definitions of $\mathcal{F}^{nec}$ and $\mathcal{O}^{nec}$ correspond to this backchaining process. Our actual

notion is slightly different: We do not start with a fact in $\mathcal{F}^{\overline{\mathcal{C}}}$ that is precondition to a path-action but with this path-action itself. Furthermore, we consider several such actions simultaneously.

Definition 3.11 (Necessary Sets of Actions and Facts)
For a planning problem $\Psi = (\mathcal{O}, \mathcal{I}, \mathcal{G})$, the functions $\mathcal{F}^{nec}_{\Psi} : 2^{\mathcal{O}} \mapsto 2^{\mathcal{F}^{\overline{\mathcal{C}}}_{\Psi}}$ and $\mathcal{O}^{nec}_{\Psi} : 2^{\mathcal{O}} \mapsto 2^{\mathcal{O}^{\overline{\mathcal{C}}}_{\Psi}}$ are defined as follows:

1. $\mathcal{F}^{nec}_{\Psi}(\mathcal{M}) = \big(\mathcal{G} \cup \bigcup_{o \in \mathcal{M}} pre(o)\big) \cap \mathcal{F}^{\overline{\mathcal{C}}}_{\Psi}$ *and*
2. $\mathcal{O}^{nec}_{\Psi}(\mathcal{M}) = \mathcal{M} \cap \mathcal{O}^{\overline{\mathcal{C}}}_{\Psi} \ \cup \ \mathcal{O}^{nec}_{\Psi}\Big(\big\{ o \bigm| f \in \mathcal{F}^{nec}_{\Psi}(\mathcal{M}),\ o \in \mathcal{O}^{\overline{\mathcal{C}}}_{\Psi},\ f \in add(o) \big\} \Big)$.

As usual, we allow the arguments to be single actions, sequences, (side)paths, and sets of (side)paths, all denoting the total set of their actions. ◇

Note that Point 1 of the previous definition restricts $\mathcal{F}^{nec}$ to be a subset of $\mathcal{F}^{\overline{\mathcal{C}}}$, which means that a dependency cannot go "through" the structured part of a domain. In other words, the backchaining process ends by reaching a fact of the structured part. By using the sets $\mathcal{F}^{nec}$ and $\mathcal{O}^{nec}$, we now can specify the dependency between the structured part and the unstructured part of a planning problem in terms of the structured part.

Definition 3.12 (The Function $\mathcal{F}^{through}$)
For a planning problem $\Psi = (\mathcal{O}, \mathcal{I}, \mathcal{G})$ and a set $\mathcal{M} \subseteq \mathcal{O}$ of actions we define the function $\mathcal{F}^{through}_{\Psi} : 2^{\mathcal{O}^{\mathcal{C}}_{\Psi}} \mapsto 2^{\mathcal{F}^{\mathcal{C}}_{\Psi}}$ as follows:

1. $\mathcal{F}_1(\mathcal{M}) = \big\{ f \bigm| o \in \mathcal{O}^{nec}_{\Psi}(\mathcal{M}),\ f \in pre(o) \cap \mathcal{F}^{\mathcal{C}}_{\Psi} \big\}$
2. $\mathcal{F}_2(\mathcal{M}) = \big\{ f \bigm| \mathcal{M}' = \mathcal{O}^{nec}_{\Psi}\big(\mathcal{M}\big), o \in \mathcal{O}^{\mathcal{C}}_{\Psi}, add(o) \cap \mathcal{F}^{nec}_{\Psi}(\mathcal{M}') \neq \emptyset, f \in add(o) \cap \mathcal{F}^{\mathcal{C}}_{\Psi} \big\}$
3. $\mathcal{F}^{through}_{\Psi}(\mathcal{M}) = \mathcal{F}_1(\mathcal{M}) \cup \mathcal{F}_2(\mathcal{M})$ ◇

If we are interested in a set $\mathcal{M}$ of path-actions, the set $\mathcal{O}^{nec}(\mathcal{M})$ contains the actions of $\mathcal{O}^{\overline{\mathcal{C}}}$ on which $\mathcal{M}$ can depend. In turn, the actions in $\mathcal{O}^{nec}(\mathcal{M})$ depend on the structured part of Ψ, because they require c-invariants to be in certain states and the execution of certain path-actions, we referred to these kinds of dependencies as type (2) and (3). Path reduction needs these dependencies as subset of $\mathcal{F}^{\mathcal{C}}$: In case of (2), these are just the preconditions of $\mathcal{O}^{nec}(\mathcal{M})$ in $\mathcal{F}^{\mathcal{C}}$, as given by $\mathcal{F}_1$.

In case of an dependency of type (3), we use the all facts $add(o) \cap \mathcal{F}^{\mathcal{C}}$ of actions o that add a precondition of an action in $\mathcal{O}^{nec}(\mathcal{M})$. These are just the facts that are collected in set $\mathcal{F}_2$. Finally, we now can express the dependency of a set $\mathcal{M}$ of path-actions on the structured part of the planning domain by the set $\mathcal{F}^{through}(\mathcal{M})$, which is the union of $\mathcal{F}_1$ and $\mathcal{F}_2$. The name $\mathcal{F}^{through}$ is hereby motivated by the fact that path reduction has to consider paths that pass through the facts in $\mathcal{F}^{through}(\mathcal{M})$. We will elaborate on the role of $\mathcal{F}^{through}$ in Chapter 5 on Page 65. The following example shows how $\mathcal{F}^{through}$ captures the dependency of path-actions on the structured part of a planning problem.

Example 3.15 (Planning Problem and Its Unstructured Part, cont.) *Recall the planning problem which was presented in Example 3.13 on Page 31. Its goal depends on the unstructured part of the problem, because* $\mathcal{F}^{nec}(\emptyset)$ *is the set* $\{g_2, g_3\}$ *and* $\mathcal{O}^{nec}_{\Psi}(\emptyset)$ *is the set* $\{o_6\}$*. Therefore action* o_6 *is necessary for every solution to this problem.*

The path-actions of the solution are $\mathcal{M} = \{o_1, o_3\}$*. The sets* $\mathcal{F}^{nec}_{\Psi}(\mathcal{M}) = \{g_1, g_2, g_3\}$ *and* $\mathcal{O}^{nec}_{\Psi}(\mathcal{M}) = \{o_5, o_6\}$ *show the dependency of these actions on the unstructured part of the problem. In turn, their dependency on the structured part is given by the set* $\mathcal{F}^{through}_{\Psi}(\mathcal{M}) = \{f_2, f_3\}$*, which means that the solution might have to pass through facts* f_2 *and* f_3*.* ■

3.1.8 The Complexity of Finding a Path of a Solution

In the previous sections we have introduced the view of plans as combination of paths, which gives rise to the idea of planning by first finding the paths of a solution and then combining these paths to a solution plan. Unfortunately, the following theorem convinces us that this approach to planning does not yield a radical improvement in terms of the overall complexity.

Theorem 3.1 (To Find a Path of a Solution is as Hard as Planning) *The general problem of finding a path through a c-invariant* C *that comprises all path-actions of* C *in a solution to a planning problem is as hard as planning.* □

Proof:

"→": Finding a path that comprises all path-actions of C in a solution to a planning problem is not harder as planning because such paths can be extracted from the solution in constant time.

"←": STRIPS planning is *PSPACE*-complete [11]. Thus, it suffices to find a polynomial reduction of a planning problem $\Psi = \big(\mathcal{O}, \mathcal{I}, \mathcal{G}\big)$ to a problem that encodes its solutions as path in a c-invariant.

The reduction extends problem Ψ by two new facts f_1, f_2 and a new action $o' = \big(\{f_2\}, \{f_1\}, \{f_2\}\big)$. The reduced problem is $\Psi' = (\mathcal{O}' \cup \{o'\}, \mathcal{I} \cup \{f_1\}, \mathcal{G} \cup \{f_1\}\big)$, where the actions in set $\mathcal{O}'$ are the actions of $\mathcal{O}$, extended to be path-actions of a new c-invariant $C = \{f_1, f_2\}$. In detail, $\mathcal{O}' = \big\{\big(pre(o) \cup \{f_1\}, add(o) \cup \{f_2\}, del(o) \cup \{f_1\}\big) \mid o \in \mathcal{O}\big\}$.

Obviously, there is a bijection between solutions to the original and the reduced problem: The $(2i-1)$-th action of a solution S' to Ψ' is the i-th action of its mapped solution to Ψ, just without facts f_1 and f_2. Each remaining action of S' is an instance of action o'. Therefore, finding a path through C that comprises all path-actions of C in a solution to Ψ' means to find a solution to Ψ. As we did not restrict our choice of Ψ, the former is at least as hard as the latter. ■

3.1.9 Normal Form of Planning Problems

We conclude our examination of c-invariants by defining a normal form of planning problems. By transforming a problem into this normal form, we compile away special cases that otherwise would complicate the definition of path reduction in the next chapters. The complexity of the transformation is a low polynomial in the size of the problem and it is neutral regarding performance, as the size and complexity of the normalized problem stays the same or is decreased.

Definition 3.13 (Normalization of Planning Problems)
Let $\Psi_1 = (\mathcal{O}_1, \mathcal{I}, \mathcal{G})$ be a planning problem and $\mathcal{C}_1$ be the set of c-invariants of Ψ_1. Then $\Psi_2 = (\mathcal{O}_3, \mathcal{I}, \mathcal{G})$ is the normalization *of problem Ψ_1 if the following holds.*

1. *$\mathcal{C}_2$ is the set of c-invariants in $\mathcal{C}_1$ that are observed by the initial state $\mathcal{I}$ of problem Ψ_1, i. e., $\mathcal{C}_2 = \{C \mid C \in \mathcal{C}_1, |C \cap \mathcal{I}| = 1\}$.*
2. *$\mathcal{C}_3$ is a subset of $\mathcal{C}_2$ such that $\bigcup_{C \in \mathcal{C}_3} C = \bigcup_{C \in \mathcal{C}_2} C$ and every c-invariant in $\mathcal{C}_3$ has a fact that is not in any other c-invariant of $\mathcal{C}_3$, i. e., for every $C \in \mathcal{C}_3$ it holds that $C \setminus \bigcup_{C' \in \mathcal{C}'_3} C' \neq \emptyset$ where $\mathcal{C}_3 = \mathcal{C}'_3 \cup \{C\}$ and $C \notin \mathcal{C}'_3$.*
3. *$\mathcal{O}_2$ is a subset of $\mathcal{O}_1$ such that $\mathcal{O}_2 = \mathcal{O}_1 \setminus \mathcal{O}'_1$ and $\mathcal{O}'_1$ is the set of all actions that are sidepath-actions of a c-invariant which is destroyed in the initial state $\mathcal{I}$ of problem Ψ_1, i. e., $\mathcal{O}'_1$ is the set $\{o \mid o \in \mathcal{O}, C \in \mathcal{C}_3, |pre(o) \cap C| \geq 1, |C \cap \mathcal{I}| = 0\}$.*
4. *$\mathcal{O}_3$ is a subset of $\mathcal{O}_2$ such that $\mathcal{O}_3 = \mathcal{O}_2 \setminus \mathcal{O}'_2$ and $\mathcal{O}'_2$ is the set of all actions in $\mathcal{O}_2$ that have two or more facts of a c-invariant as precondition, i. e., $\mathcal{O}'_2 = \{o \mid o \in \mathcal{O}_2, C \in \mathcal{C}_3, |pre(o) \cap C| \geq 2\}$.*

Furthermore, we say that $\mathcal{C}_3$ is the set of c-invariants of Ψ_2. ◇

The rationales behind the points of the previous definition are as follows: The first point excludes c-invariants that are violated or destroyed by the initial state. We remove these c-invariants because path reduction is not amenable to them (cf. Chapter 5). Point 2 is to prevent planning problems with a number of c-invariants that is exponential in the number of facts. See Section 3.1.5 on Page 27 for further details. The last two points eliminate actions which are not applicable in any state that is reachable from the initial state; in other words, that are not part of any solution.

All c-invariants of a normalized planning problem Ψ are observed by the initial state. In other words, there is no need to consider violated c-invariants in further sections. Furthermore, every action of a normalized planning problem has either a single fact of a c-invariant among its precondition or it is not a sidepath-action of that c-invariant, so that every action in Ψ is potentially applicable.

The following theorem assures us that every solution to a normalized planning problem is a solution to its initial problem and vice versa.

Theorem 3.2 (Solutions of Normalized Planning Problems)
Let $\Psi' = (\mathcal{O}, \mathcal{I}, \mathcal{G})$ be the normalization of planning problem Ψ. Then S is a solution to Ψ' if and only if S is a solution to Ψ. □

Proof: Every action of Ψ' is an action of Ψ, thus if S is a solution to Ψ' then S is a solution to Ψ, too. For the opposite direction, we observe that no action in $\mathcal{O}'_1$ and $\mathcal{O}'_2$ is applicable in any state reachable from the initial state $\mathcal{I}$. Therefore, all actions in S are in $\mathcal{O}_3$ and S is a solution to problem Ψ', too. (q. e. d.)

From now on, we solely consider normalized planning problems. Therefore, we can stipulate the following simplification: $post(o) \cap C$ is either the empty set or the set $\{f\}$, i. e., a set of size one. In case we know that the set is non-empty, we say that $post(o) \cap C$ is the fact f. In other words, we refer to the set $\{f\}$ by its sole element f. Likewise, we say f is the fact $pre(o) \cap C$ if $pre(o) \cap C = \{f\}$.

3.2 Replaceable Sequences of Actions

Planning problems tend to have numerous solutions and many of them are similar. Often, such similar solutions differ only by a reordering of actions that do not interfere with each other. Then, we only have to regard one of these solutions, the other can be discarded. More generally, a sequence of actions may be *replaceable* by a different sequence, in the sense that whenever the first sequence occurs, the second can be substituted.

Definition 3.14 (Replacement and Segmentation of Sequences)
Let T be a plan. A segmentation *of T is a list $\ldots, T_i, \ldots$ of sequences such that T equals the concatenation $\ldots \circ T_i \circ \ldots$ The sequences in $\mathcal{S}$ are called* segments *of T.*

If $\mathcal{S}$ is a segmentation of a plan T and T_i is the i-th segment of $\mathcal{S}$ then the replacement *of T_i in T by a sequence T'_i yields the sequence $\ldots \circ T_{i-1} \circ T'_i \circ T_{i+1} \circ \ldots$* ◇

We are interested in replaceable action sequences in two ways. First of all, path reduction is based on the replacements of paths and we will explain features of path replacement by comparison with the replacement of sequences. Secondly, the replacement of paths has to replace the unstructured part of paths, too, which is done according to the ordinary replacement of sequences. Let us give an example.

Example 3.16 (Blocks World)
The well-known blocks world formalizes stacks of blocks on a table. The domain has three predicates schemata, on(?x, ?y), clear(?x), and on-table(?y), which state that block ?x is on top of block ?y, there is no block on top of ?x, and block ?y is placed on the table, respectively. Hereby, ?x and ?y are variables that can be bound by a block. For example, the fact on(A, B) denotes that block A is on top of block B. Note that in the blocks world different variables of a schema have to be bound by pairwise distinct blocks.

The blocks world comprises the following action schemata: move(?x, ?y, ?z), move-onto(?x, ?y), and move-from(?x, ?y). The first denotes moving block ?x from block ?y on top of block ?z, i. e., deleting fact on(?x, ?y) and adding fact on(?x, ?z). Similarly, the second and the third action schema denotes moving block ?x from block ?y onto the table and moving block ?x from the

table onto block ?y, respectively. The following table gives the full definition of the blocks world action schemata.

action	*preconditions*	*added facts*	*deleted facts*
move(?x, ?y, ?z)	*clear(?x), clear(?z), on(?x, ?y)*	*on(?x, ?z), clear(?y)*	*clear(?z), on(?x, ?y)*
move-onto(?x, ?y)	*clear(?x), on(?x, ?y)*	*on-table(?x), clear(?y)*	*on(?x, ?y)*
move-from(?x, ?y)	*clear(?x), clear(?y), on-table(?x)*	*on(?x, ?y)*	*clear(?y), on-table(?x)*

Many sequences in the blocks world can be replaced by other sequences. An example of such a replaceable pair in the blocks world domain is $\langle move(A, B, D), move(A, D, C)\rangle$ *and* $\langle move(A, B, C)\rangle$*: Whenever a block is moved twice in a row, this sequence can be replaced by a single move directly to the destination of the second move. This is also an example where replaceability is applicable in one direction only: Replacing the second sequence by the first may result in an invalid plan if block D is covered by another block.* ■

Let us now give a formal definition of the replacement of sequences.

Definition 3.15 (Replaceability of Sequences)
A sequence T_1 *is* replaceable *by a sequence* T_2 *if*

1. *Sequence* T_2 *is conflict-free,*
2. $pre(T_2) \subseteq pre(T_1)$*,*
3. $post(T_1) \setminus pre(T_1) \subseteq post(T_2)$*, and*
4. $del(T_2) \subseteq del(T_1)$*.* ◇

If T_1 *is a partial sequence of a plan* T*, i. e.,* $T = T' \circ T_1 \circ T''$*, then the* exchange *of* T_1 *by a sequence* T_2 *in* T *yields the sequence* $T' \circ T_2 \circ T''$*. The* context *of* T_1 *in* T *is the sequence* $T' \circ T''$*.* ◇

We call a sequence replaceable by another if in every solution the exchange of the first by the second yields another solution. Another notion of replaceability is based on a generation and test approach: Let $T \circ T_1$ and $T \circ T_2$ be two solution prefixes. Furthermore, let the execution of $T \circ T_1$ and $T \circ T_2$ in the initial state yield the state s_1 and s_2, respectively. Then T_1 is replaceable by T_2 if s_2 entails s_1. This notion of replaceability helps for example in forward search by reducing the number of candidate solutions. As it refers to a particular state, it does not allow to find pairs of sequences that are replaceable in general.

Theorem 3.3 (Exchange of Replaceable Sequences is Solution Preserving)
Let Ψ *be a planning problem and* S *be a solution to* Ψ*. If* T *is a subsequence of* S *and* T' *is a sequence that can replace* T *then the exchange of sequence* T *through sequence* T' *in solution* S *yields a solution to problem* Ψ*.* □

Proof: Let Ψ be the planning problem $\Psi = \big(\mathcal{O}, \mathcal{I}, \mathcal{G}\big)$ and S be the sequence $T_1 \circ T \circ T_2$. Let the execution of T_1 in $\mathcal{I}$ yield state s_1. Because of $pre(T') \subseteq pre(T)$ we know that sequence T' is applicable in state s_1. Furthermore, we know that T' is conflict-free.

Let the execution of T and T' in s_1 yield the states s_2 and s_2', respectively. Then s_2 state is a subset of state s_2': Let f be a fact that is in state s_2. Let us first assume $f \in s_1$. Then $f \notin del(T)$, which implies $f \notin del(T')$ and therefore fact f is in state s_2'. If otherwise $f \notin s_1$ then $f \in post(T)$ and $f \notin pre(T)$. Therefore $f \in post(T')$ and fact f is in state s_2', too.

From $s_2 \subseteq s_2'$ follows that sequence T_2 is applicable in state s_2'. Let the execution of sequence T_2 in states s_2 and s_2' yield the states s_3 and s_3'. Then we know with the previous argument that state s_3 is a subset of state s_3' because every conflict-free sequence is replaceable by itself. As S is a solution to problem Ψ, the goal $\mathcal{G}$ is a subset of state s_3. Therefore, the goal $\mathcal{G}$ is a subset of s_3', too, and $T_1 \circ T' \circ T_2$ is a solution to problem Ψ. (q. e. d.)

With c-invariants and the replacement of sequences we now have the foundation of path replacement, which we introduce in the next chapter.

Chapter 4

Path Replacement

The last chapter introduced c-invariants, an important class of domain knowledge. The c-invariants of a planning domain allow, for example, to classify and compare solutions to a planning problem because plans of the problem interfere with these c-invariants and all solutions have related interferences. In particular, the more similar such interferences, namely paths and sidepaths, are the more likely is their equivalence in respect to plans. In the following, we use this observation to develop path replacement, a technique based on paths that can replace each other in plans.

4.1 Introduction

Intuitively, the view of planning domains as collection of c-invariants suggests a planning strategy via divide and conquer: For each c-invariant find a path that reaches its accepting state, then combine these separate paths into one solution plan. Unfortunately, c-invariants are not amenable to this simple idea: The execution of a path in one c-invariant has side-effects on other c-invariants. Such sidepaths complicate the composition of arbitrary paths to a conflict-free sequence. In fact, the general problem of finding a single path through a c-invariant that is part of a solution is as hard as planning itself, as shown in Section 3.1.8 on Page 34.

Instead of using paths for planning directly, we propose their use in a reduction technique. The idea is to relate paths according to their replaceability in plans and to accept paths as relevant that cannot be replaced. We then exclude paths from solutions that are not identified as relevant. Hereby, a path is replaceable if for every plan that has the first path there is another plan that differs from the first solely by its use of the second path instead of the first. Before we go into details, let us give an example of replaceable paths.

Example 4.1 (Blocks World)
Recall the blocks world domain as given in Example 3.16 on Page 36, which is a simple yet non-trivial domain. It is highly structured by c-invariants and many paths in the blocks world are replaceable.

The blocks world has two c-invariants for each block, which correspond to the facts that each block is in exactly one location and is either covered by exactly one block or is free. We abbreviate the c-invariants for a block $?x$ the following way: $C^l_{?x}$ shall denote the set of facts that correspond to possible

locations of block ?x and $C^c_{?x}$ corresponds to the possible covers of block ?x. For example, in a blocks world domain with blocks A, B, and C, the c-invariant C^l_A denotes the set $\{on(A,B), on(A,C), on\text{-}table(A)\}$ and the c-invariant C^c_B the set $\{on(A,B), on(C,B), clear(B)\}$.

For the blocks world it is well known that it is not necessary to move a block more than twice: It is always possible to move it onto the table and from there directly to its final destination. Therefore, the maximal length of the shortest solution to a blocks world problem with n blocks is $2n$.

For example, instead of the sequence $T_1 = \langle move(A,B,C), move\text{-}onto(A,C), move\text{-}from(A,D)\rangle$ we can always use the sequence $T_2 = \langle move\text{-}onto(A,B), move\text{-}from(A,D)\rangle$. Both sequences are paths in the c-invariant C^l_A and the path (T_2, C^l_A) can always replace the path (T_1, C^l_A) under the segmentation $\big(\langle move\text{-}onto(A,B)\rangle, \langle\,\rangle, \langle move\text{-}from(A,D)\rangle\big)$: If sequence T_1 is in a path, we find the corresponding plan simply by replacing the first action of sequence T_1 by the first action of sequence T_2, removing the second action of T_1, and keeping the last action of sequence T_1 as the last of sequence T_2. ■

Our motivation behind the use of path replacement is to improve the performance of planning systems. Which properties do we expect from the replacement of paths in this respect? In short, we want (1) many paths to be replaceable, (2) that replaceable paths can be exchanged in many plans, and (3) that the decision is easy whether the exchange of a replaceable path yields a valid plan. A notion of path replacement with all these properties would lead to a significantly reduced search space, thus to a simple and powerful reduction technique.

These criteria do also apply to other reduction techniques based on action sequences. For example, consider the technique based on replaceable sequences of actions, or short RSA, which has been introduced in Section 3.2 on Page 36. This technique considers only conflict-free *sub*sequences of plans that are always replaceable. Hence, it considers a only small number of sequences as candidates. In particular, it does not consider the sequences which are mentioned in the previous example because these are not subsequences but partial sequences of a plan. On the other hand, all replacements that RSA accepts are applicable as soon as a replaceable sequence is identified, i. e., regardless of the context. In summary, RSA yields only a small number of possible replacements, which, on the other hand, are always applicable with little cost.

The design of path replacement is not specified yet, so how do our options for it look like? The first design criterion states that the number of replaceable paths should be large, so that we have a large potential of reduction. The more similar the paths have to be, the fewer paths are replaceable. Therefore, we want to pose as few conditions on similarity as possible, for example by abstracting from the facts that sidepaths pass through. The ideal of applicability, the second point, means that if we call a path replaceable by another, we want the replacement to be valid in every plan, i. e., regardless of the context of the replacement. As we will see later, our notion of replaceability does not meet this ideal in full: Some dependency on the context remains and we have to check whether a replacement is valid in a specific context.

In the following, we first give the necessary properties of paths to be replaceable. In Section 4.4 on Page 48 we discuss trade-offs between options that are still open and

chose the one that balances our design criteria. We then show that if all conditions of path reduction on pairs of paths are satisfied then the exchange of one by the other in a solution yields another solution.

4.2 Comparable Paths

Before we can replace paths we have to decide whether this replacement is valid; in other words, we have to compare paths. In order to give a precise definition of this comparison, we need a corresponding terminology. Let us begin with an overview over these terms before we define them formally in subsequent sections.

We call pairs of paths *comparable* if they satisfy basic conditions for being candidates for replacement. Comparable paths are said to *correspond* if they fulfill all conditions related to ordinary c-invariants, i. e., without endangerment. The *exchange* of a path by a corresponding one is the mere removal of the actions of the first and the subsequent insertion of the actions of the second. Therefore, path exchange can yield conflicting sequences. A path that can be safely exchanged with its corresponding path in respect to fact endangerment is said to be *as safe as* the replaced one. Finally, the term *replaceable* is reserved for pairs of paths that fulfill all conditions necessary to preserve solutions under their exchange. We reserve the term *path replacement* for the exchange of paths that we call replaceable. Now let us begin the definition of path replacement by the formal definition of comparable paths and their exchange.

Definition 4.1 (Comparable Paths)
Let P_1 and P_2 be two paths through the same c-invariant; let $\mathcal{S}$ be a segmentation of P_2. We call P_2 comparable *to P_1 under $\mathcal{S}$ if the following holds:*

1. *The number of segments in $\mathcal{S}$ equals the length of P_1.*
2. *For each segment T in $\mathcal{S}$ and each c-invariant C it holds that T is conflict-free and $|pre(T) \cap C| \leq 1$.*
3. *Each segment T that destroys a c-invariant C has an action that destroys C.* $\diamond$

The rationale behind Point 2 of the definition of comparable paths is to prohibit trivial cases in which the exchange of path P_1 by path P_2 yields a conflicting sequence or a sequence that is inapplicable. For example, if $|pre(T) \cap C| = 2$ is true for a segment T and a c-invariant C then there exists no reachable state in which T can be applied. Here, the reachability of a state means reachable from an initial state of a planning problem in normal form, as given in Section 3.1.9 on Page 35.

The purpose of Point 3 is solely to simplify the definitions and proofs of path replacement. It states that if $\text{destroy}(T_i) \cap C$ is non-empty for a segment T_i in $\mathcal{S}$ then there is an action o in T_i for which $\text{destroy}(o) \cap C$ is non-empty, too. Hence, it does not consider segments that destroy c-invariant C without having an action that destroys C. We can formulate Point 3 less restrictive by demanding $\text{destroy}(T_i) \subseteq \text{destroy}(o_i)$, where o_i is the i-th action of P_1 and T_i is the i-th segment of P_2, in which case we say that o_i and T_i *correspond*. This alternative formulation of identifies more paths as replaceable but requires the recalculation of $\text{destroy}(T)$ for every new segment T. We leave such an extension to further work.

Definition 4.2 (Exchange of Comparable Paths)
Let T be a plan, P_1 a path in T, and P_2 be a path that is comparable to P_1 under segmentation $\mathcal{S}$. The exchange *of P_1 by P_2 in T is the substitution of the i-th action of P_1 in T by the i-th segment of P_2, for all $1 \leq i \leq n$. The* context *of the exchange is the sequence of all actions in T that are not in P_1.* ◇

The exchange of a path by a comparable path in a plan yields a new sequence. Hereby, we differentiate three kinds of actions: Those of the exchanged path in the original plan, those in segments of the new path, and those in the context of the exchange. Actions of the latter kind are therefore in both sequences: the original plan and the resulting sequence. The reader may note that this notion of exchange of a path is similar to the notion of exchange of a sequence as given in Definition 3.15 on Page 37.

If two paths are comparable under a given segmentation then each action of the one path is associated with a segment of the second. The following definition introduces a notation for the relation between actions of the two paths.

Definition 4.3 (Relations between Actions in Comparable Paths)
Let P_2 be a path that is comparable to path P_1 under a segmentation $\mathcal{S}$. Let o_1 be the i-th action of path P_1 and T be the j-th segment in segmentation $\mathcal{S}$. We say that action o_1 and segment T correspond *if $i = j$.*

Let o_2 be an action in segment T. Then we say that o_1 and o_2 are simultaneous *if o_1 and T correspond. We say that action o_1 is* before *action o_2 if $i < j$ and that action o_1 is* after *action o_2 if $i > j$.* ◇

We use a similar notion for subsequences of these paths. Note that the notion of an action lying during a sequence is introduced in Definition 2.6 on Page 8.

Definition 4.4 (Relations between Sequences in Comparable Paths)
Let P_2 be a path that is comparable to path P_1 under a segmentation $\mathcal{S}$. Let T_1 be a subsequence of path P_1 and o_2 an action in path P_2. Under segmentation $\mathcal{S}$, we say that action o_2 lies during *sequence T_1 if it is simultaneous with an action in T_1. Likewise, we say that action o_2 is* before *and* after *T_1 if o_2 is before the first action of T_1 and after the last action of T_1, respectively. Let T_2 be a subsequence of path P_2. Under segmentation $\mathcal{S}$, we say that sequence T_2 lies* during *sequence T_1 if the first and the last action of sequence T_2 lie during sequence T_1.* ◇

These definitions allow us to relate the features of comparable paths in a convenient way, e. g., to examine whether a sidepath of one of them begins earlier than a sidepath of the other. Let us give an example.

Example 4.2 (Terminology of Path Replacement)
We draw the motivation for the terminology of Definitions 4.3 and 4.4 from their temporal relation in a parallel execution of both sequences: the original plan and the new sequence yielded by the replacement of a path. Hereby, a time step allows to either execute an action of the context or an action of the original path together with its corresponding segment in the new sequence.

Let us now demonstrate the use of the terminology in the following situation: $T_1 = \langle o_1, o_2, o_3, o_4 \rangle$ is a plan and $P_1 = \big(\langle o_1, o_3, o_4 \rangle, C\big)$ is a path in plan T_1.

Furthermore, $P_2 = \big(\langle o'_1, o'_2, o'_3\rangle, C\big)$ *is a path and* $\mathcal{S} = \big(\langle o'_1, o'_2\rangle, \langle o'_3\rangle, \langle\,\rangle\big)$ *is a segmentation of* P_2*. Under this segmentation, path* P_2 *can be compared to path* P_1*. The exchange of path* P_1 *by path* P_2 *in plan* T_1 *yields the sequence* $T_2 = \langle o'_1, o'_2, o_2, o'_3\rangle$ *and the context of this exchange is the sequence* $\langle o_2\rangle$*. The following figure shows the ordering relation of actions in sequences* T_1 *and* T_2*.*

T_1:	o_1	o_2	o_3	o_4
T_2:	o'_1, o'_2	o_2	o'_3	

Regarding the relative position of actions and subsequences in these two sequences we can say the following: Action o_1 *lies during path* P_1 *in plan* T_1 *and action* o_2 *is after action* o_1 *and before action* o_3 *in path* P_1*. Under the segmentation* $\mathcal{S}$*, actions* o_1 *in plan* T_1 *and action* o'_2 *in sequence* T_2 *are simultaneous. Action* o'_3 *in sequence* T_2 *is after action* o_1 *and before action* o_4 *in plan* T_1*. Subsequence* $\langle o'_1, o'_2\rangle$ *of sequence* T_2 *lies during subsequence* $\langle o_1\rangle$ *of plan* P_1*, whereas this is not true for subsequence* $\langle o'_2, o_2\rangle$*.* ■

Let us make the following convention for reasons of simplicity: From now on, if we refer to a single segmentation, i. e., an arbitrary but fixed one, then we do not reference the segmentation. In other words, we say "regarding a segmentation $\mathcal{S}$" only if we consider two or more segmentations at the same time.

Now we are ready to describe the properties of replaceable paths. First, we introduce the notion of correspondence between sidepaths.

4.3 Correspondence of Sidepaths and of Paths

The exchange of a replaceable path in a plan should yield another conflict-free plan. For this to be true, the sidepaths of the replaced and the replacing paths have to be similar, we say they have to *correspond*. For example, if a replaced path has a sidepath that connects two different facts then the replacing path has to connect the same facts. Otherwise, the precondition of an action in the path is false and the resulting sequence has a conflict. Likewise, a sidepath of the replacing path can only connect two facts if the replaced path has a similar sidepath. In other words, there has to be a mapping between the sidepaths of the two paths such that mapped sidepaths correspond.

In this section, we examine the mapping between the sidepaths of comparable paths and, in particular, the restrictions on the location of the beginning and ending of mapped sidepaths. We then use this mapping to define the notion of correspondence of paths.

4.3.1 Correspondence of Sidepaths, the Common Case

It is not sufficient for mapped sidepaths to connect the same facts. For example, if a replacing sidepath begins before the replaced one then an action of the context can introduce a conflict. The following example shows conditions in which the exchange of comparable paths can yield a conflicting sequence.

Example 4.3 (Elongation of Sidepaths and Cores Yields Conflicts)
Our aim is to find conditions under which a sidepath can safely replace another sidepath in every context. The first example shows that the replacing sidepath has to begin simultaneously or later than the replaced sidepath. Let us consider the following situation: P_1 is a path in a plan T and P_2 is a path that is comparable to P_1. Q_1 and Q_2 are sidepaths of P_1 and P_2, respectively, that go through the same c-invariant and that connect the same facts. If sidepath Q_2 begins before sidepath Q_1 then an action of the context T' can introduce a conflict, as we see in the following figures. Hereby, the left figure shows plan T before the exchange of path P_1 and the right one after the exchange.

The next figure shows a situation in which the exchange of a path P_1 by a path P_2 in T yields a conflict even though sidepath Q_2 begins simultaneously with Q_1. The reason is that the core of Q_2 begins before the one of Q_1, which results in a different active fact between the segments during the fringe of Q_1. This change of the active fact then conflicts with an action of the context. Again, the left figure shows plan T before the replacement of path P_1 and the right one after the replacement.

■

The following definition of correspondence combines all conditions that are necessary for pairs of regular sidepaths.

Definition 4.5 (Correspondence of Sidepaths with Regular Ending)
Let P_1 a path and P_2 a path that is comparable to P_1 under a segmentation $\mathcal{S}$. Let Q_1 be a sidepath of path P_1 with regular ending. Let Q_2 be a sidepath of path P_2. Then sidepaths Q_1 and Q_2 correspond *under segmentation $\mathcal{S}$ if they satisfy the following statements:*

1. *Sidepath Q_2 is regular and connects the same facts through the same c-invariant as sidepath Q_1.*

2. *Sidepath Q_2 lies during sidepath Q_1.*

3. *If action o is in P_1 and lies during a fringe of sidepath Q_1 then* $\mathrm{persist}(T, C)$ *is true, where T is the segment that corresponds to o. Furthermore, if o is in a fringe of sidepath Q_1 then $pre(T) \cap C \subseteq pre(o) \cap C$ is true.*

Hereby, persist *is a function that indicates whether or not the execution of a sequence leaves the active fact of a c-invariant unchanged, i. e.,* $\mathrm{persist} : \mathcal{O}^* \times \mathcal{C}_\Psi \mapsto$ boolean *and* $\mathrm{persist}(T, C) =_{def} \big(pre(T) \cap C = post(T) \cap C\big)$. ◇

Note that the definition of correspondence does not mention the core of sidepath Q_2. In fact, the core of the replacing sidepath can begin earlier or end later than the core of the replaced one. Nevertheless, Point 3 of this definition guarantees that an action in a fringe of Q_1 is always replaced by a partition T for which persist(T, C) is true, i. e., that is persistent in respect to the active fact of the sidepath. In other words, even if the core of Q_2 begins before the core of Q_1, the additional path-actions result in the same active facts between the segments. Therefore, this point prevents situations as the one given in the second case of Example 4.3 on the facing page.

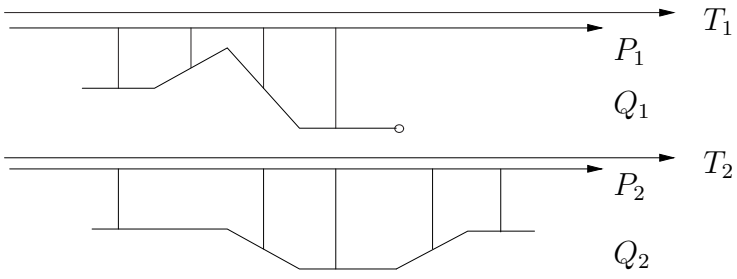
Also note that Point 3 demands $pre(T) \cap C = \emptyset$ for each segment T that replaces an action o if this action lies during a fringe of sidepath Q_1 but is not in that sidepath. This means that segments during a fringe of Q_1 can only have an action of Q_2 if their corresponding action is in Q_1. Again, this condition prevents potential conflicts with the context of the replacement.

4.3.2 Correspondence of Sidepaths, the Case of a Dead End

The preceding definition introduces correspondence solely for sidepaths with regular endings; now we handle sidepaths with dead end. Here we can be less restrictive as for sidepaths with regular ending: If a sidepath has a dead end then no action after its end is a sidepath-action of its c-invariant, in particular no action of the context. Furthermore, if it is in a solution to a planning problem then the goal does not mention an active fact of its c-invariant. Therefore, a replacing sidepath can have an arbitrary end, i. e., it can either end with an arbitrary fact or have a dead end. We see such a case in the following example.

Example 4.4 (Correspondence of Sidepaths with Dead End)
The replacement of sidepaths with dead end require weaker conditions than the replacement of sidepaths with regular ending. Let us give an example: P_1 is a path in a plan T_1 and P_2 is a path that is comparable to P_1. Q_1 and Q_2 are sidepaths of P_1 and P_2, respectively, that go through the same c-invariant and that begin with the same facts. They differ in their ending: Q_1 has a dead end and Q_2 has a regular end. The exchange of P_1 by P_2 in T_1 yields the sequence T_2.

The following figure shows plan T_1 before and after the replacement. With the vertical placement of both sequences we visualize the temporal ordering between, e. g., the beginnings of the exchanged sidepaths.



Despite the fact that sidepath Q_2 ends later than Q_1, the replacement of P_1 does not introduce a conflict: Q_1 has a dead end, so there is no sidepath-action

of C in T_1 after Q_1. Consequently, the additional actions of Q_2 cannot conflict with the context. ■

The following definition sums up all requirements on the replacement of a sidepath with dead end.

Definition 4.6 (Correspondence of Sidepaths with Dead End)
Let P_1 a path and P_2 a path that is comparable to P_1 under a segmentation $\mathcal{S}$. Let Q_1 be a sidepath of path P_1 through c-invariant C with dead end. Let Q_2 be a sidepath of path P_2. Then sidepaths Q_1 and Q_2 correspond *under segmentation $\mathcal{S}$ if they satisfy the following statements:*

1. *Sidepath Q_2 goes through c-invariant C and begins with the same fact as sidepath Q_1. Furthermore, if Q_1 has a core then $post(T) \cap C \subseteq \{f\}$, where T is the longest prefix of Q_2 that ends before or simultaneous with the core of Q_1 and fact f is the end of Q_1.*

2. *Sidepath Q_1 begins before or simultaneously with sidepath Q_2. Furthermore, if sidepath Q_2 has a dead end then it ends simultaneously with or after sidepath Q_1.*

3. *Let o be an action in P_1 and T be its corresponding segment of path P_2. If action o lies during a fringe of sidepath Q_1 then $pre(T) \cap C \subseteq pre(o) \cap C$ is true. Furthermore, if o is in a fringe of Q_1 and before the last action of sidepath Q_1 then* $\text{persist}(T, C)$ *is true.* ◇

Definition 4.6 poses fewer restrictions on the replacement of sidepaths with dead end than Definition 4.5 on Page 44 on the replacement of sidepaths with regular ending. In detail, Point 1 allows the replacing sidepath Q_2 to be longer than the replaced one. If Q_2 has a dead end as well, then Point 2 requires it to end simultaneously with or later as Q_1. Hereby, we prevent that the destruction occurs earlier in the new plan as in the old.

Point 1 states that if sidepath Q_1 has a core then $post(T) \cap C \subseteq \{f\}$, where T is the longest prefix of sidepath Q_2 that ends before or simultaneous with the core of Q_1, Q_1 goes through c-invariant C, and fact f is the end of Q_1. This is to guarantee that Q_1 and Q_2 pass through the same fact during the right fringe of Q_1. In Definition 4.5 of correspondence for paths with regular endings we said the same by simply stating that sidepaths Q_1 and Q_2 connect the same facts, i. e., end with the same fact. We cannot do this here, as the end of Q_2 is arbitrary.

Finally, Point 3 poses fewer restrictions on the segment T that corresponds to the last action of sidepath Q_1: T is allowed change the active fact of the sidepath. As there is no sidepath-action of c-invariant C in the context after the end of Q_1 there is no need for T to persist the active fact of C.

We have now covered all cases of correspondence between sidepaths. But what if there is no corresponding sidepath? The next section elaborates on this case.

4.3.3 Correspondence of Sidepaths: Disappearing Sidepaths

The previous definitions of correspondence require that every sidepath of a replacing path has a corresponding sidepath in the replaced path. This does not hold the other

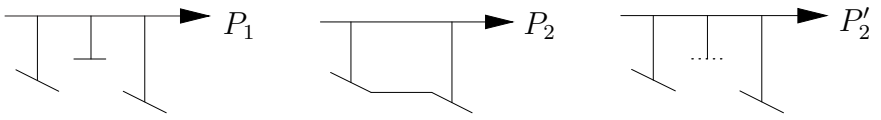
way around: A circular sidepath begins and ends with the same fact, in other words, it preserves the active fact of its c-invariant. As replacing sidepaths can be shorter than the sidepaths they replace, a circular sidepath can be replaced by a sidepath of length zero. Thus, a circular sidepath can be exchanged by the empty sequence; in other words, it can disappear.

Example 4.5 (Disappearing Sidepaths)
Reconsider the blocks world domain of Example 4.1 on Page 39. There, we have seen that path $P = \big(\langle \textit{move-onto}(A, B), \textit{move-from}(A, D)\rangle, C^l_A\big)$ *can always replace path* $P' = \big(\langle \textit{move}(A, B, C), \textit{move-onto}(A, C), \textit{move-from}(A, D)\rangle, C^l_A\big)$. *Path* P' *has a circular sidepath* Q *through* C^c_C *which specifies the coverage of block* C. *Path* P *is independent of this block and consequently does not have a sidepath through* C^c_C. *Therefore, the replacement of path* P' *by path* P *in a plan means to remove sidepath* Q *from the plan.* ■

Disappearing sidepaths have an interesting consequence: A circular sidepath can separate two other sidepaths that go through the same c-invariant. If the first of them ends with the same fact as the second begins, the disappearance of the circular sidepath "joins" them: Definition 3.7.4 on Page 26 states that sidepaths are as long as possible.

Example 4.6 (A Disappearing Sidepath Joins Two Others)
Let us consider the following replacement in which a disappearing sidepath joins two other sidepaths. Path P_1 *has three sidepaths through a c-invariant* C. *The first sidepath ends with the same fact as the third begins and the second one is circular.*

Path P_2 *has the same actions as path* P_1 *with the difference that the actions of the second sidepath of path* P_1 *are no longer sidepath-actions of c-invariant* C. *In other words, path* P_2 *is path* P_1 *with the second sidepath removed. Surprisingly, the sidepaths of path* P_2 *do not correspond to the sidepaths of path* P_1*: Path* P_2 *has only one sidepath through c-invariant* C *but two, as shown by the middle figure.*



■

Joining sidepaths can also result from endangering actions whose replacing segment is not endangering (cf. Definition 3.7.3 on Page 26). Cascading joinings and disappearances of sidepaths are possible, too. All such cases can be included into our notion of correspondence by the introduction of an "empty" sidepath, whose sole purpose is to separate sidepaths of a replacing path that otherwise would join. This way, the resulting sidepaths could correspond to sidepaths of a replaced path. The right figure of the previous example depicts such an empty sidepath as dotted line. For sake of simplicity, we leave such an extension of path replacement for further work.

4.3.4 Correspondence of Paths

We now define a related notion of correspondence for paths. Note that each path is its own sidepath, so the comparability of sidepaths applies to paths, too.

Definition 4.7 (Correspondence of Paths)
Let P_1 and P_2 be two paths. We say that P_2 corresponds *to P_1 under a segmentation $\mathcal{S}$ if the following holds:*

1. *P_2 is comparable to P_1 under $\mathcal{S}$.*

2. *There is a injective mapping between sidepaths of paths P_2 onto those of P_1 such that mapped sidepaths correspond. Furthermore, every sidepath of P_1 that is not mapped is circular.* ◇

In the following, we prove the properties of path replacement by arguing over the respective location of sidepaths in the two involved paths. Hereby, we expect every action and every segment to have at most one sidepath through a given c-invariant. Let us assure ourselves of this fact.

Lemma 4.1 (Uniqueness of Sidepaths Regarding Actions and Segments)
Let P_2 be a path that corresponds to a path P_1 under a segmentation $\mathcal{S}$. Then (1) every action in path P_1 lies during at most one sidepath of path P_2 through a given c-invariant. Likewise, (2) every action in path P_2 lies during at most one sidepath of path P_1 through a given c-invariant. □

> **Proof:** (1) Let o be an action in path P_1. Path P_2 is comparable to P_1, thus Definition 4.1.2 on Page 2 states that $|pre(T) \cap C| \leq 1$, where the action o corresponds to segment T and C is a c-invariant. Then we know with Definition 3.7.2 on Page 26 that all sidepath-actions of C in T belong to the same sidepath. (2) Every action in path P_1 is in at most one sidepath through a given c-invariant, thus the conjecture trivially holds. (q. e. d.)

Note that the disappearance of sidepaths is the reason why the mapping of sidepaths in Definition 4.7 of path correspondence is an injection. If the number of sidepaths is the same in both paths, i. e., no sidepath disappears then the mapping is a bijection.

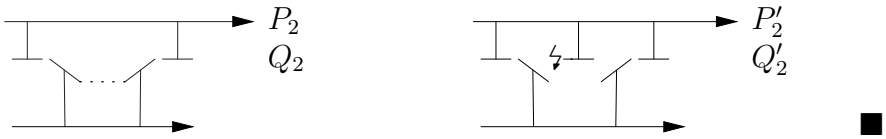
4.4 Free and Bound Sidepaths

With the current notion of correspondence we relate the fringes of comparable sidepaths. But what about their cores? In this section, we will see that path replacement does not restrict the core of a replacing sidepath. Instead, we restrict the applicability of a replacement to those cases in which the context cannot interfere with the core.

Recall that our first design goal of path replacement was to have a large number of replaceable paths. This means that we want the similarity between corresponding sidepaths to be small. The definition of correspondence adheres to this idea by leaving the replacement of the core unspecified. In other words, the subsequence of the replacing path that lies during the core of the replaced path is arbitrary. But we have to pay a price for this flexibility. Let us give an example.

Example 4.7 (Interference Between a Core and Its Context)
The exchange of a path can interfere with its context. Consider path P_1 in the following figure, whose core is interleaved by a sidepath-action of the context. If we exchange P_1 by path P_1', the resulting sequence has a conflict.

In theory, the same could happen with the fringes of sidepaths: In the following figure, P_2 is a path with a coreless sidepath Q_2. P_2' is the same path as P_2, with the difference that its sidepath Q_2' has an additional sidepath-action, i. e., it has a sidepath-action in a segment whose corresponding action in P_2 is not an action in Q_2. The exchange of P_2 by P_2' results in a conflict because this additional sidepath-action interferes with the context. Our definition of correspondence prevents this problem: Points 3 of Definitions 4.5 on Page 44 and 4.6 on Page 46 state that Q_2 and Q_2' do not correspond.



■

The exchange of a path can introduce a conflict because of its context: If an action in the context is a sidepath-action of a c-invariant C and lies during the core of a sidepath through C then the exchange of the sidepaths path can change the active fact of C before the context action. Hence, we have to prohibit the exchange of paths if the context can interfere. In this case, we say that the context *binds* the path and otherwise that it is *free*.

Definition 4.8 (Free and Bound Paths, Free and Bound Sidepaths)
Let P be a path in a plan T, Q be a sidepath of P through c-invariant C, and o an action in T. We say that action o binds *sidepath Q if o is a sidepath-action of c-invariant C, o lies during the core of sidepath Q in path P, and o is not in Q. We call sidepath Q* free *in plan T if Q is not bound in T. We call path P* free *in plan T if all sidepaths of P are free in T.* ◇

Recall our three design goals for path replacement: (1) Large number of replaceable paths, (2) wide applicability, and (3) easy application. The case of bound sidepaths is the reason why these goals are mutually exclusive to some extend: To focus on the first criterion means to depend on the context whereas the last two require independence of context. In the following, we discuss trade-offs between different options of path replacement, and find a definition that balances them.

If we want to maximize the number of replaceable paths then we have to abstract from the active facts that a sidepath passes through, besides the facts it connects. In other words, we have to disregard context. This defies criterion (2) and (3) because the replacement is likely to fail in many contexts and whether or not it fails has to be checked for every pair of an action and its corresponding segment.

Thus, independence of context requires that if a fact f of a c-invariant is in the post-condition of a segment then this fact has to be in the postcondition of the corresponding

action. Furthermore, if an action of a path is not sidepath-action of a c-invariant then the same has to hold for its corresponding segment. Both conditions are entailed by $pre(T) \cap C \subseteq pre(o) \cap C$, which is restrictive and defies the design criterion (1).

Another possibility is to simply check whether context has interleaving actions. If not then the replacement is allowed. With this approach there is still a large number of replaceable paths and whether their exchange yields a conflict-free plan is easy to verify. On the other hand, the number of cases where the exchange can actually be performed is small, so it still defies (2).

Our solution is a mixed approach: We abstract from the induced path of a sidepath, hence we do not allow interleaving actions during core. For the fringes, we require $pre(T) \cap C \subseteq pre(o) \cap C$, so they can be interleaved by the context. To know whether a replacement is valid in a specific context, we simply check for interleaving actions during the core. If there are such actions then the path is bound and the exchange is not valid; otherwise the path is free and can be safely replaced. The previous Example 4.7 on Page 48 shows both cases: The sidepath of path P_1 is bound because a sidepath-action lies during its core; sidepath Q_2 is free because sidepaths need to have a core to be bound. The following lemma states this observation more generally.

Lemma 4.2 *Coreless sidepaths, as well as sidepaths with a core of length one, are free.* □

> **Proof:** A bound sidepath has a path-action before and after the binding action, so its core is of length two or more. (q. e. d.)

4.5 Endangerment and Destruction of c-Invariants

Unfortunately, it is not safe in all cases to replace paths whose sidepaths correspond. In addition to correspondence, we have to guarantee that the new path does not destroy more c-invariants than the old as well as that the context does not destroy the c-invariants of the sidepaths of the new path.

> **Example 4.8 (Path Exchange in the Presence of Endangerment)**
> *In the following, we demonstrate endangerment by three examples of sidepaths that go through endangered c-invariants, so that their exchange result in a conflict. The first figure shows a sidepath that is endangered by an action o during its core: Path P is conflict-free because the endangered fact is not active during action o. After exchanging path P by path P′, the corresponding sidepath has the endangered fact active, so the resulting sequence has a conflict.*

The second figure shows a similar situation: Here, the sidepath is circular and has a core whereas the corresponding sidepath is coreless. Action o endangers the fact which is connected by the sidepaths. Again, the exchange of path P

through path P' yields a conflict. As a consequence, we cannot allow circular sidepaths with core to disappear if they begin with an endangered fact.

The last figure shows that an exchanged path can yield a conflict if it endangers an additional fact. Hereby, sidepath Q is in the context and is not altered by the exchange of path P by path P'.

■

Whether or not the exchange of a path in a plan is safe regarding endangerment depends on the path that is exchanged. We define the term *as safe as* for this relation between paths.

Definition 4.9 (A Path is as Safe as Another Path)
Let P_1 be a path and let P_2 be a path that corresponds to P_1 under a segmentation $\mathcal{S}$. Then path P_2 is as safe as *path P_1 if for all actions o in P_1 the following is true. Hereby, T is the segment in $\mathcal{S}$ that corresponds to o, C is a c-invariant of Ψ, Q_1 is a sidepath of P_1 through C, and Q_2 is a sidepath of P_2 that corresponds to Q_1.*

1. *If segment T endangers a fact f of c-invariant C and action o does not endanger that fact, i. e.,* $\text{endanger}(T) \cap C \not\subseteq \text{endanger}(o)$*, then one of the following is true:*
   (a) *o is either in Q_1 or it lies during the core of Q_1. Furthermore, one of the following is true:*
      i. *T is before Q_2 (or Q_1 disappeared) and f is not the beginning of Q_1.*
      ii. *T lies during Q_2.*
      iii. *T is after Q_2 and f is not the ending of Q_1.*
   (b) *Q_1 has a dead end and o is either the last action of Q_1 or after Q_1.*
2. *One of the following is true:*
   (a) *The following two conditions hold:*
      i. *Sequence T has at most the facts of $\mathcal{F}_\Psi^{end} \cap C$ as postcondition that are postcondition of action o. In other words, $post(T) \cap \mathcal{F}_\Psi^{end} \cap C \subseteq post(o)$.*
      ii. *If the precondition $pre(o) \cap C = \{f\}$ of action o is in $\mathcal{F}_\Psi^{end}$ and o is a path-action of C then the active fact of C in S_2 after T is different from f.*
   (b) *Q_1 has dead end, o is the last action of Q_1 or later, and the last action of Q_2 lies in T.*

Point 1 of the preceding definition states that actions of P_2 can endanger arbitrary facts of c-invariant C if they lie during the core of a sidepath through C of P_1. That is because no action in the context during that core is a sidepath-action of C.

Action o in Point 2(a)ii is a path-action of c-invariant C. It changes the active fact of C, so that fact f is not a postcondition of o. Note that this implies if Q_1 begins with a fact of $\mathcal{F}^{end}$ and it has a core then this core cannot disappear. Together with Point 2(a)i this means that Q_2 cannot pass through a fact in $\mathcal{F}^{end}$ if Q_1 does not simultaneously pass through the same fact. The restrictions of Point 2a are unnecessary if Q_1 has a dead end and the last action of Q_2 lies in T. In this case there are no sidepath-actions of C after T in S_2, thus a possible destruction of C cannot yield a conflict.

4.6 Replacing the Unstructured Part of Paths

By now we solely covered facts of c-invariants. Finally, we show how to replace the unstructured part of sequences (cf. Section 3.1.7 on Page 31), i. e., the part comprised by facts that are not in any c-invariant. For this, we use the replacement of action sequences as introduced in Section 3.2 on Page 36. Of course, we only want to use the replacement of sequences for facts in $\mathcal{F}^{\overline{\mathcal{C}}}$ but not for those in $\mathcal{F}^{\mathcal{C}}$, so we limit sequences to those fact.

Definition 4.10 (Replacement of a Sequence Regarding a Fact Set)
Let limit *be a function that maps a pair of an action sequence and a set of facts onto an action sequence, i. e.,* $\text{limit} : \mathcal{O}^* \times 2^{\mathcal{F}} \mapsto \mathcal{O}^*$. *The semantics of* limit *is* $\text{limit}(\langle \dots, o_i, \dots \rangle, \mathcal{F}) = \langle \dots, o_i', \dots \rangle$, *where* $o_i' = \big(pre(o_i) \setminus \mathcal{F}, add(o_i) \setminus \mathcal{F}, del(o_i) \setminus \mathcal{F}\big)$.

We say that a plan T_1 is replaceable by an action sequence T_2 regarding a fact set $\mathcal{F}$ if T_1' is replaceable by T_2', where $\text{limit}(T_1, \mathcal{F}) = T_1'$ *and* $\text{limit}(T_2, \mathcal{F}) = T_2'$. ◇

The replacement of action sequences regarding a fact set behaves right as expected.

Theorem 4.3 (Replacement of Limited Action Sequences)
Let Ψ be a planning problem and T be a plan of Ψ. Let $\mathcal{F} \subseteq \mathcal{F}_\Psi$ a set of facts. Let $T = T_1 \circ T_2 \circ T_3$. Let T_2' be the sequence that can replace T_2 regarding $\mathcal{F}$. Then the sequence $\text{limit}(T_1 \circ T_2' \circ T_3, \mathcal{F})$ *is conflict-free.* □

> **Proof:** $\text{limit}(T_1 \circ T_2' \circ T_3, \mathcal{F})$ is conflict-free if the same is true for $\text{limit}(T_1, \mathcal{F}) \circ \text{limit}(T_2', \mathcal{F}) \circ \text{limit}(T_3, \mathcal{F})$. The latter follows directly from Theorem 3.3 on Page 37. (q. e. d.)

Enough spadework for now, let us go on with the definition of path replacement and prove that it is solution preserving.

4.7 Path Replacement

We now combine the properties given in the previous sections to a definition of path replaceability.

Definition 4.11 (The Replacement of Paths)
Let Ψ be a planning problem. Let P_1 and P_2 be a paths of Ψ and let $\mathcal{S}$ be a segmentation of P_2. Then path P_2 can replace *path P_1 under segmentation $\mathcal{S}$ if the following holds:*

1. *Paths P_1 and P_2 correspond.*
2. *Path P_2 is as safe as path P_1.*
3. *Every segment T in $\mathcal{S}$ can replace its corresponding action o in P_1 regarding $\mathcal{F}_{\Psi}^{\overline{C}}$.*

We say path P_2 can replace *path P_1, i. e., without mentioning a segmentation, if there exists a segmentation under which P_1 can replace path P_2. We call the exchange of a path P by a path P' in a plan T a* replacement *if P' can replace P under $\mathcal{S}$.* $\diamond$.

The remainder of this section shows that the replacement of paths in a solution always yields another solution.

4.7.1 Path Replacement Without Endangerment

We show the properties of path replacement by induction over conflict-free prefixes of the resulting sequence. Hereby, we start with a simplified notion of path replacement, i. e., a version that disregards fact endangerment by actions. The definition of a simplified replacement situation allows to write the corresponding lemmas in a more convenient way. Note that we deviate from our convention of using "T" to denote plans because this allows to reduce the number of subscripts in the upcoming theorems and proofs.

Definition 4.12 (Replacement Situations)
A replacement situation *is a six-tuple $\big(\Psi, S_1, S_2, P_1, P_2, \mathcal{S}\big)$, where Ψ is a planning problem, S_1 is a plan of Ψ, P_1 is a path in S_1, P_2 is a path of Ψ, and $\mathcal{S}$ is a segmentation of P_2. Furthermore, under segmentation $\mathcal{S}$, P_2 corresponds to path P_1, P_2 is as safe as P_1, and the exchange of path P_1 by path P_2 in plan S_1 yields the sequence S_2.* $\diamond$

We first show a lemma that provides the induction step for our proof for the validity of path replacement. In short, the lemma says that if a prefix T_2 of S_2 is conflict-free then either $\mathit{post}(T_2) \cap C = \emptyset$ or $\mathit{post}(T_2) \cap C = \mathit{pre}(o) \cap C$ is true for all c-invariants C, where o is the action that follows T_2 in S_2. To show this property, we examine the last sidepath-action o' of C in T_2, if it exists.

The proof is organized as a large case differentiation: In particular, each of o and o' can be either in the context or in P_2. In addition, we differentiate between three different locations for an action in a sidepath: The left fringe of the sidepath, its core, and its right fringe. Similarly, there are four different ways for an action to lie during a sidepath but not to be in it: during its left fringe, during its core, during its right fringe, and before or after the sidepath; in other words, the action does not lie during the sidepath. The first two cases are simple:

1. There is no sidepath-action of C in T_2; in other words, action o' does not exist. Clearly, this case implies $\mathit{post}(T_2) \cap C = \emptyset$.

2. If actions o' and o are either both in the context and they are not separated by an action of path P_1 or they are both in the same segment then we know $post(o') \cap C = pre(o) \cap C$ because plan S_1 and the segments of path P_2 are conflict-free.

In the main cases action o' exists and is separated from o: Either both actions are in the context and are separated in S_1 by an action of P_1, both actions are in different segments of path P_2, or one of them is in the context and the other is in a segment. If actions o and o' lie during the same sidepath through C then we have the following sub-cases:

3. Both action o' and o lie during the same sidepath Q_1 through C of P_1. Note that an action o'' in P_1 that lies during the fringe of Q_1 can only be replaced by a segment that has an action of Q_2 if o'' is in Q_1, so that o and o' are separated by an action of Q_2 only if their corresponding actions in S_1 are separated by an action of Q_1. Furthermore, Q_1 is free and all sidepath-actions of C in S_1 during the core of Q_1 are in Q_1.

   (a) o is either before the core of Q_1 or simultaneously with the first action of this core.

   (b) o' is before the core of Q_1 and o is after this core.

   (c) o' is simultaneously with an action of the core of Q_1 but not the last action of the core.

   (d) o is simultaneously with an action of the core of Q_1 but not the first action of the core.

   (e) o' is either simultaneously with the last action of the core of Q_1 or after this core.

The four remaining cases are as follows:

4. Actions o' and o lie during two different sidepaths through C in P_1.

5. Action o' does not lie during a sidepath through C in P_1 and o lies during sidepath Q_1 through C in P_1.

6. Action o' lies during sidepath Q_1' through C in P_1 and o does not lie during a sidepath through C in P_1.

7. Both actions o' and o are in the context and none of them lies during a sidepath through C in P_1.

These cases are exhaustive for the locations of actions o' and o in S_2.

In the upcoming proofs, we often encounter the case that two sidepath-actions o' and o of a c-invariant C in the new sequence S_2 lie during a sidepath Q through C of the original path P_1 and these actions are the last one before and first one after a sidepath-action of Q, respectively. Then we are interested in the active fact of C between o' and o in S_2 and expect that $post(o') \cap C = pre(o) \cap C$. The following lemma assures us that our intuition

is correct by relating the pre- and postcondition of these actions to the beginning and ending of Q_1.

Note that if o is an action in sequence S_2 then the term "action o lies during sidepath Q_1 in S_1" means either one of the following: (1) Action o is in the context and therefore it lies in sequence S_1 during Q_1 but it is not in sidepath Q_1. If otherwise action o is in sidepath Q_2 of P_2 then o is in a segment T_i of segmentation $\mathcal{S}$ and the corresponding action o_i in S_1 can either be (2) in Q_1 or (3) lie during but not be in Q_1.

Lemma 4.4 (About the Active Fact During the Fringes of Sidepaths)
Let $(\Psi, S_1, S_2, P_1, P_2, \mathcal{S})$ be a replacement situation. Furthermore, Q_1 is a sidepath of P_1, o_1 is an action in Q_1, and o_2 is an action in S_2 during Q_1. Then the following conjectures hold.

1. *If o_2 is the last sidepath-action of C before o_1 in S_2 and*
   (a) *if o_1 is before the core of Q_1 or the first action of this core then $post(o_2) \cap C$ is the beginning of Q_1.*
   (b) *if o_1 is after the core of Q_1 then $post(o_2) \cap C$ is the ending of Q_1.*
2. *If o_2 is the first sidepath-action of C after o_1 in S_2 and*
   (a) *if o_1 is before the core of Q_1 then $pre(o_2) \cap C$ is the beginning of Q_1.*
   (b) *if o_1 is the last action of the core of Q_1 or after this core then $pre(o_2) \cap C$ is the ending of Q_1.* □

Proof: We show point (1a) as follows: If o_1 is before the core of Q_1 or it is the first action of this core then o_2 is during the left fringe of Q_1. If o_2 is in the context then it is in S_1, too. Furthermore, it is the last action in the context before an action o_1' that is in the left fringe of Q_1 or the first action of the core of Q_1. Therefore, $post(o_2) \cap C = pre(o_1') \cap C$. If otherwise o_2 is in Q_2 then it is the last sidepath-action in a segment T. As the corresponding action o in S_1 is in the left fringe of Q_1, we know that $post(T) \cap T = post(o) \cap C$. In both cases, $post(o_2) \cap C$ is the beginning of Q_1. The other three cases follow from a similar argument. (q. e. d.)

Now, let us continue with the induction hypothesis for the replacement without endangering actions.

Lemma 4.5 (Without Endangerment the Exchange is Conflict-Free)
Let $(\Psi, S_1, S_2, P_1, P_2, \mathcal{S})$ be a replacement situation. Furthermore, no action in P_2 endangers a fact of a c-invariant. Let $T_2 \circ \langle o \rangle$ be a prefix of sequence S_2. Let $pre(o) \cap C = \{f\}$, where C is a c-invariant of planning problem Ψ. Then the following conjecture is true:

If T_2 is conflict-free and no action in T_2 endangers fact f then either there is no sidepath-action of C in T_2 or for the last sidepath-action o' of C in T_2 it holds $post(o') \cap C = \{f\}$. □

Proof: We differentiate between the cases regarding the location of o' and o in T_2 that are enumerated on Page 53. In particular they can be either in P_2 or in the context, and they can be in different locations during a sidepath. Each of the cases has either to imply $post(T_2) \cap C = \emptyset$, to imply $post(o') \cap C = \{f\} = pre(o) \cap C$, or to be inconsistent.

1. No action in T_2 is a sidepath-action of C, which implies $post(T_2) \cap C = \emptyset$. In all following cases there is a last sidepath-action o' of C in T_2 with $post(o') \cap C = \{f\}$.
2. Actions o' and o in S_1 are in the context and all actions between o and o' are in the context, too. Then $post(o') \cap C = pre(o) \cap C$ follows from the conflict-freeness of S_1. If o' and o are in the same segment then the same is true because paths P_1 and P_2 are comparable, i. e., all segments in $\mathcal{S}$ are conflict-free (cf. Definition 4.1 on Page 41). In all following cases, if both actions o and o' are in the context then they are separated in S_1 by an action of P_1.
3. Both actions o' and o lie during the same sidepath Q_1 through C in P_1. Here, we determine the active fact of C before and after o' and o according to Lemma 4.4 on the preceding page and differentiate the following sub-cases.
   (a) o' lies during the left fringe of Q_1 and o is either before the core of Q_1 or it is simultaneously with the first action of the core of Q_1. Then we know that both facts $post(o') \cap C$ and $pre(o) \cap C$ are the beginning of Q_1, thus they are the same.
   (b) o' is before the core of Q_1 and o is after the core of Q_1. Then fact $post(o') \cap C$ is the beginning of Q_1 and fact $pre(o) \cap C$ is the ending of Q_1. As all actions in the core of Q_1 are replaced by segments without sidepath-action of C, Q_1 is circular and $post(o') \cap C = pre(o) \cap C$.
   (c) o' lies during the core of Q_1 but not simultaneously with its last action. Therefore, o lies during the core of Q_1, too. As Q_1 is free, actions o' and o are not in the context and therefore are adjacent action in Q_2. Then $post(o') \cap C = pre(o) \cap C$ follows from the definition of sidepaths.
   (d) o lies during the core of Q_1 but not simultaneously with its first action. Here, we use the same argument as in the preceding case.
   (e) o' is either simultaneously with the last action of the core of Q_1 or after the core. Furthermore, o lies during the right fringe of Q_1. Then we know that both facts $post(o') \cap C$ and $pre(o) \cap C$ are the ending of Q_1, thus they are the same.
4. Actions o' and o lie during two different sidepaths Q_1 and Q_2 through C in P_1, respectively. This case yields a contradiction.

   If there are sidepaths in P_1 through C between Q_1 and Q_1' then they disappear, hence they are circular. Furthermore, there is no action in the context between o and o', so $post(o) \cap C$ is the ending of Q_1 and $pre(o') \cap C$ is the beginning of Q_1'. Finally, Q_1 and Q_1' are not separated by an action

that endangers a fact of C. Then we know from the definition of sidepaths that Q_1 and Q_1' are the same sidepath, which is a contradiction.

5. Action o' does not lie during a sidepath through C in P_1 and o lies during sidepath Q_1 through C in P_1.

   If there are sidepaths in P_1 through C between o and Q_1' then they disappear, hence they are circular. Furthermore, there is no action in the context after o and before Q_1, hence $post(o') \cap C$ is the beginning of Q_1.

   All actions of Q_1' between o' and o disappear. If o is during the core of Q_1' then o is in Q_1', so $pre(o) \cap C$ is the beginning of the core, i. e., the beginning of Q_1'. If o is during the left core of Q_1' then $pre(o) \cap C$ follows from Point 2b of Lemma 4.4 on Page 55, if o is after the core then $pre(o) \cap C$ follows from Point 2a of this lemma. Therefore, $pre(o) \cap C$ is the beginning of Q_1, too.

6. Action o' lies during sidepath Q_1' through C in P_1 and o does not lie during a sidepath through C in P_1. Here, we use the same argument as in the preceding case.

7. Both actions o' and o are in the context and none of them lies during a sidepath through C in P_1. Furthermore, there is an action of P_1 between o and o'.

   If there is no sidepath through C in P_1 between o' and o then $post(o') \cap C = pre(o) \cap C$. Otherwise, there is a sidepath Q_1'' through C in P_1 between o' and o, i. e., Q_1'' begins after o' and ends before o. As there is no sidepath-action of C in the context between o' and o, the beginning of Q_1'' is $post(o') \cap C$ and its ending is $pre(o) \cap C$. Furthermore, there is no sidepath of P_2 that corresponds to Q_1'', thus Q_1'' is circular. In other words, $post(o') \cap C = pre(o) \cap C$.

These cases are exhaustive, thus we have shown that either there is no sidepath-action of C in T_2 or $post(o') \cap C = pre(o) \cap C$, where o' is the last sidepath-action of C in T_2. (q. e. d.)

4.7.2 Path Replacement Including Endangerment of Facts

We continue by showing that fact endangerment by single actions does not invalidate the result of the preceding section, i. e., that the induction hypothesis is also true for replacement with actions that endanger a fact. We split our argumentation in the case that an endangering action is in the context and that it is in path P_2.

Lemma 4.6 (Path Replacement and Endangerment in the Context)
Let $(\Psi, S_1, S_2, P_1, P_2, \mathcal{S})$ be a replacement situation. Let $T_2 \circ \langle o \rangle$ be a prefix of sequence S_2. Furthermore, no action in P_2 endangers a fact of a c-invariant. Then the following conjecture is true:

If T_2 is conflict-free then no action o^* in T_2 endangers a fact f that is active during o^*. Hereby, $pre(o) \cap C = \{f\}$ and C is a c-invariant of planning problem Ψ. □

Proof: In other words, T_2 does not destroy f. We show the lemma by contradiction. Let us assume that there is an action o^* in T_2 that endangers the fact f. Then either there is no sidepath-action of C in T_2 at all or the last one, we call it o', is before o^*. In other words, if action o' exists then o^* is between o' and o. Otherwise, o^* can be any action in S_2. Note that if o' is after o^* then T_2 does not destroy f because T_2 is conflict-free.

First let us make the following two remarks: (1) o^* is an action in the context because the prerequisites state that no action in path P_2 endangers a fact. (2) In this proof we frequently refer to Definition 4.9 on Page 51. Point 2a of this definition states that two properties hold for each segment T that includes an action of a sidepath Q_2, except if Q_1 has a dead end. This exception can only apply if o' and o lie in the same segment of P_2, which implies that o^* cannot lie between o' and o. Therefore Point 2a holds for all such segments T.

There is a path-action o_1 of C in S_1 before o^* with f as precondition. Let us assume otherwise. If there is no path-action of C before o in S_1 at all then $f \in \text{endanger}(T_1)$, where T_1 is the longest prefix of S_1 during T_2. Furthermore, $pre(S_2) \cap C = \{f\}$ because all segments of $\mathcal{S}$ in S_2 before o persist in C and none of the actions in the context before o is a path-action of C. As $pre(S_1) \cap C = pre(S_2) \cap C$, this contradicts that S_1 is a solution to Ψ.

If the first path-action o_1 of C in S_1 is between o^* and o then f is its precondition: As there is no sidepath-action of C in S_2 between o^* and o, in particular no path-action of C, we know that o_1 is in P_1. Therefore, P_1 has a sidepath through C and $pre(o_1) \cap C = pre(o) \cap C$. This contradicts that S_1 is conflict-free.

Therefore, there are path-actions of C in S_1 before o^*. Then our assumption implies that f is not precondition of any path-action of C before o^* and $pre(S_1) \cap C \neq \{f\}$. In contrast, f is active in S_2 before o^*, thus there is a path-action o'_3 before o^* in P_2 with $post(o'_3) \cap C = \{f\}$. There is no such action in S_1 (it is conflict-free), so that o'_3 is not in the context but in a segment T of P_2. Let o_3 be the action in S_1 that corresponds to T. As f is a fact in $\mathcal{F}_\Psi^{end}$, we know with Point 2(a)i of Definition 4.9 that f is a postcondition of o_3, too, which is a contradiction.

Therefore, there are path-actions of C before o in S_1 that have f as postcondition. Let o_1 be the last hereof.

There is a path-action o_2 of C in T_2 before o^*. Let us assume otherwise. Then o is either an action in the context, an action in the left fringe of a sidepath through C in P_2, or the first action of the core of that sidepath. In all these cases there has to be a last path-action o_3 of C in S_1 between o^* and o such that both o_1 and o_3 are in the same sidepath of P_1 through C. The core of this sidepath encloses o^* and prevents the destruction of C by changing the active fact of C before o^* to a fact different than f. Furthermore, the first path-action o'_3 of this core has f as precondition. Let T be the segment in $\mathcal{S}$ that corresponds to o'_3. Then we know with Point 2(a)ii of Definition 4.9 that there is a path-action of C in T, which is a contradiction.

In other words, there are path-actions of C in T_2 before o^*. Let o_2 be the last hereof.

All orderings between o_1 and o_2 yield a contradiction. If o_1 is after o_2 then o_1 is in P_1; otherwise o_1 is in S_2, too, and o_2 is not the last path-action of C in T_2 before o^*. As f is a postcondition of o_2, we know with Point 2(a)ii of Definition 4.9 that there is a path-action of C in T, where T is the segment that corresponds to o_1, which contradicts our assumption that o_1 is after o_2.

If o_1 is before o_2 then Point 2(a)i of Definition 4.9 states that action o_3 in S_1 has the same postcondition as o_2, i. e., $post(o_3) \cap C = \{f\}$, where o_3 is the action in P_1 that is simultaneous to o_2. There is no path-action in S_1 between o_3 and o^* because it would have to have f as precondition. Therefore, f is the active fact in S_1 before o^*, which is a contradiction because S_1 is conflict-free.

The only case left is that o_1 and o_2 are simultaneous. In case o_1 is in P_1 then we have the preceding case with $o_1 = o_3$. The case that o_1 is in the context yields a contradiction because $post(o_2) \cap C = \{f\} \neq post(o_1) \cap C$.

All cases contradict our assumption that there is an action o^* in T_2 that endangers the fact f, where o^* is after the last sidepath-action of C in T_2, thus no such action exists and the lemma holds. (q. e. d.)

The preceding Lemma 4.6 shows that actions in the context do not endanger fact f; we now show the same for actions in P_2. Definition 4.9.1 on Page 51 states that endanger$(T_i) \subseteq$ endanger(o_i) for each action o_i of S_1 and its corresponding segment T_i, unless they satisfy certain conditions. These exceptions relate an additional endangerment of T_i to a sidepath Q_1 of P_1 and its corresponding sidepath Q_2 in P_2: If T_i endangers a fact f that is in c-invariant C and o_i does not endanger f, then T_i lies during sidepath Q_1 through c-invariant C in P_1, during the sidepath Q_2 in P_2 that corresponds to Q_1, or both. Hence, we can show the validity of the additional endangerments by sole examination of Q_1 and Q_2.

Lemma 4.7 (Path Replacement and Endangerment in the Replacing Path) *Let $(\Psi, S_1, S_2, P_1, P_2, \mathcal{S})$ be a replacement situation. Let $T_2 \circ \langle o \rangle$ be a prefix of sequence S_2. Then the following conjecture is true: If T_2 is conflict-free then no action o_2^* in T_2 endangers a precondition of o that is active during o_2^*.* □

Proof: In other words, sequence T_2 does not destroy a precondition of o. Let $pre(o) \cap C = \{f\}$, where C is a c-invariant of planning problem Ψ. From Lemma 4.6 on Page 57 we know that no action in the context endangers f. In other words, if an action in T_2 endangers f then it is an action in a segment of P_2.

We divide the proof into two parts: First, we consider the case that a segment T^* of $\mathcal{S}$ in T_2 endangers fact f and o_1^* does not endanger f. Hereby, o_1^* is the action in S_1 that corresponds to T^*. Then, the second part covers the case where f is endangered by T^* and by o_1^*. In both cases, the endangerment of f by T^* is caused by an action o_2^*, i. e., action o_2^* is in T^* and endangers f.

Part 1: Segment T^* endangers fact f and action o_1^* does not. In other words, $f \in \text{endanger}(o_2^*) \setminus \text{endanger}(o_1^*)$. Then Definition 4.9.1 states that T^*, and therefore o_2^*, is related to a sidepath Q_1 through C of P_1 and its corresponding sidepath Q_2 in P_2. Note that if o_1^* is in Q_1, then the active fact in S_1 before o_1^* is known.

If o_2^* is in Q_1 and it is before Q_2 (or Q_1 disappeared, i. e., there is no such sidepath Q_2) then we know that the active fact of C in S_2 during o_2^* is the beginning of Q_1. Point 1(a)i of Definition 4.9 prevents this fact from being endangered. Point 1(a)iii guarantees the same if o_2^* is after Q_2.

From Definition 3.7.3 of sidepaths we know that no action in P_2 endangers the active fact of C during Q_2. For this reason, Point 1(a)ii of Definition 4.9 allows the endangerment of arbitrary facts of C by segments during Q_2. Still, we have to restrict o_2^* to be simultaneous with an action of Q_1 during the fringes of Q_1 because the active fact of C is not specified between adjacent actions of fringes. This is unnecessary during the core of Q_1: Q_1 is free, so there is no potentially conflicting sidepath-action of C during its core.

Finally, we have the case of Q_1 having a dead end and o_2^* being either the last action of Q_1 or after Q_1. Here, C is destroyed in S_1 before or simultaneously with o_2^* and there are no sidepath-action of C in after o_2^*. Hence, the endangerment of f by o_2^* cannot yield a conflict.

Part 2: Segment T^* and action o_1^* both endanger fact f. In other words, $\text{endanger}(o_2^*) \subseteq \text{endanger}(o_1^*)$. Note that if o_1^* lies during a sidepath Q_1 of P_1 then it does not endanger the active fact of Q_1, which is prohibited by the definition of sidepaths.

Therefore, if o_1^* lies during the fringes of Q_1 then T^* can endanger the same facts as o_1^* because the active fact of Q_1 persists during fringes. If o_1^* lies during the core of Q_1 then Definition 4.9.2 on Page 51 guarantees that Q_2 passes through an endangered fact of C only if Q_1 passes through the same fact, where Q_2 is the sidepath of P_2 that corresponds to Q_1. As S_1 is conflict-free, the endangerment of T^* cannot cause a conflict. If T lies after Q_1 then this sidepath has a dead end and the endangerment of a fact of C by T cannot cause a conflict, too.

What is left to show is the case where o_2^* does not lie during such a sidepath Q_1 nor after Q_1 if it has a dead end. Let T_1 be the longest prefix of S_1 that is during T_2. Let s_1 and s_2 the states that are yielded by the execution of T_1 and T_2 in $\mathcal{I}$, respectively. Then $s_1 \cap C = s_2 \cap C$, so that the lemma follows from $\text{endanger}(T^*) \subseteq \text{endanger}(o_1^*)$.

Both parts of the proof imply the lemma, therefore it holds. (q. e. d.)

4.7.3 Path Replacement

We are now ready to proof that the replacement of a replaceable path in a solution yields another solution. We first show that path replacement yields conflict-free plans by

induction over conflict-free prefixes of the resulting sequence. Hereby, we use the induction hypothesis which is developed in the preceding sections.

Lemma 4.8 (Path Replacement Yields Conflict-Free Plans)
Let $\big(\Psi, S_1, S_2, P_1, P_2, \mathcal{S}\big)$ be a replacement situation. Then S_2 is a conflict-free plan. □

> **Proof:** We show the lemma by induction over the length n of a prefix T_2 of sequence S_2. In the base case $n = 0$ the sequence T_2 is empty and therefore conflict-free.
>
> Induction hypothesis: If T_2 is a conflict-free prefix of S_2 with length n then the prefix of S_2 with length $n + 1$ is conflict-free, too. Let o be the action that succeeds T_2 in S_2. Then the induction hypothesis is true if for every fact f in $pre(o)$ it holds that either one of $f \in post(T_2)$ or $f \notin del(T_2)$ is true.
>
> If f is in $\mathcal{F}_\Psi^{\mathcal{C}}$ then Lemma 4.5 on Page 55 states that either $post(T_2) \cap C = \emptyset$ or $post(T_2) \cap C = \{f\}$. Furthermore, Lemma 4.7 on Page 59 states that T_2 does not endanger f, thus from $post(T_2) \cap C = \emptyset$ follows $f \notin del(T_2) \cap C$.
>
> If f is in $\mathcal{F}_\Psi^{\overline{\mathcal{C}}}$ then Lemma 4.3 on Page 52 guarantees that $\text{limit}\big(T_2 \circ \langle o \rangle, \mathcal{F}^{\overline{\mathcal{C}}}\big)$ is conflict-free, which implies that either $f \in post(T_2)$ or $f \notin del(T_2)$.
>
> As $T_2 \circ \langle o \rangle$ is conflict-free in all cases, the induction hypothesis is true and therefore the lemma holds. (q. e. d.)

The lemma guarantees that the exchange of a replaceable path yields a conflict-free plan. We now have to prove that their exchange preserves solutions, too. We show this by reducing a planning problem to another that has two additional actions, one for its initial state and one for its goal, such that each solution to the reduced problem is a solution to the original one if these two actions are removed. Thus if the exchange of a path is conflict-free in a solution to the reduced problem then it yields a solution to the original one.

Theorem 4.9 (Path Replacement is Solution Preserving)
Let $\Psi = \big(\mathcal{O}, \mathcal{I}, \mathcal{G}\big)$ be a planning problem, S_1 be a solution to planning problem Ψ, and P_1 be a path in solution S_1. Let P_2 be a path that can replace path P_1. Then the replacement of path P_1 by path P_2 in solution S_1 yields a solution S_2 to planning problem Ψ. □

> **Proof:** We prove the theorem by reduction of Ψ to a planning problem Ψ'. We construct Ψ' in a way that there is a bijection between the solutions to Ψ and the solutions to Ψ' for which holds: If S_1 is a solution to Ψ, S_1' is a solution to Ψ', and S_1 and S_1' are mapped then S_1 is the subsequence of S_1' that results from removing the first and the last action of S_1'. Therefore, if P_1 is replaceable by P_2 in S_1 then we can replace P_1 by P_2 in S_1'. The construction of Ψ' guarantees that whenever the resulting sequence S_2' is conflict-free then it is a solution to Ψ'. Therefore, the subsequence S_2 of S_2' is a solution to Ψ.
>
> The reduction is as follows: Ψ' is the planning problem $\big(\mathcal{O} \cup \{o_\mathcal{I}, o_\mathcal{G}\}, \mathcal{I}', \{f_\mathcal{G}\}\big)$, where $o_\mathcal{I}$ is an action that has a new fact $f_\mathcal{I}$ as precondition and consumes a new fact f_C for every c-invariant C of planning problem Ψ. Furthermore action $o_\mathcal{I}$ adds all facts in $\mathcal{I}$ and a fact f_S. Action $o_\mathcal{G}$ has all facts of $\mathcal{G}$ as

precondition, adds fact $f_{\mathcal{G}}$, and consumes f_S. The initial state $\mathcal{I}'$ is comprised by the union of $\{f_{\mathcal{I}}\}$ and $\{f_C \mid C \in \mathcal{C}_\Psi\}$.

$o_{\mathcal{I}}$ and $o_{\mathcal{G}}$ are the first and the last action of all solutions to Ψ', respectively, and that this is their only occurrence in these solutions. Furthermore, $\langle o_{\mathcal{I}}\rangle \circ S \circ \langle o_{\mathcal{G}}\rangle$ is a solution to Ψ' if and only if S is a solution to Ψ. Finally, the replacement of P_1 by P_2 in a solution to Ψ' does not involve $o_{\mathcal{I}}$ and $o_{\mathcal{G}}$, and if the resulting sequence is conflict-free then it is a solution to Ψ', too. (q. e. d.)

From now on, we only exchange a path by another path if the second can replace the first.

As final property we show that path replacement is right cancelable. In other words, if the prefixes of two paths are replaceable and their remaining postfixes are the same then they are replaceable, too.

Theorem 4.10 (Path Replacement is Right Cancelable)
If a path P_1' can replace a path P_1 then the path $P_1' \circ P_2$ can replace the path $P_1 \circ P_2$. □

Proof: Let l_1 be the length of P_1, l_2 the length of P_2, and $\mathcal{S}_1$ the segmentation under which P_1' can replace P_1. Let $\mathcal{S}_2$ be the segmentation where the first l_1 segments are the same as in $\mathcal{S}_1$ and the $(l_1 + i)$-th segment, $1 \leq i \leq l_2$, is $\langle o_i\rangle$, where o_i is the i-th action of P_2.

P_1' is as safe as P_1 and every segment in $\mathcal{S}_1$ can replace its corresponding action in P_1 regarding $\mathcal{F}^{\overline{C}}$. As these segments and their corresponding action stay the same, and path P_2 can replace itself, $P_1' \circ P_2$ is as safe as $P_1 \circ P_2$ and the former can replace the latter regarding $\mathcal{F}^{\overline{C}}$.

It is left to show that $P_1 \circ P_2$ and $P_1' \circ P_2$ correspond. Obviously, $P_1 \circ P_2$ and $P_1' \circ P_2$ are comparable under $\mathcal{S}_2$. Regarding correspondence, the only interesting case is if two sidepaths merge, i. e., if a sidepath Q_1 of P_1 and a sidepath Q_2 of P_2 result in a sidepath $Q_1 \circ Q_2$ of $P_1 \circ P_2$. In case of a disappearing sidepath, i. e., if there is no sidepath Q_1' of P_1' that corresponds to Q_1 then we set Q_1' to $\big(\langle\,\rangle, C\big)$, where Q_1 is a sidepath through C. Then $Q_1 \circ Q_2$ corresponds to $Q_1' \circ Q_2$:

1. $Q_1 \circ Q_2$ and $Q_1' \circ Q_2$ go through the same c-invariant and begin with the same fact. Furthermore, they end with the same fact and are either both regular or they have both a dead end.

2. $Q_1' \circ Q_2$ lies during $Q_1 \circ Q_2$.

3. If an action of $P_1 \circ P_2$ lies during a fringe of $Q_1 \circ Q_2$ then it either lies during a fringe of Q_1 in P_1 or of Q_2 in P_2. As P_1' and P_2 can replace P_1 and P_2, respectively, the third point of correspondence is satisfied, too.

The remaining mappings are those of the unmerged sidepaths of P_1' and P_2. As the remaining unmapped sidepaths in $P_1 \circ P_2$ disappear in P_1', they disappear in $P_1' \circ P_2$, too. Therefore, $P_1 \circ P_2$ and $P_1' \circ P_2$ correspond. (q. e. d.)

A similar proof shows that path replacement is left cancelable.

Chapter 5

Path Reduction

This chapter develops a reduction strategy based on paths and their replacement. Given a planning problem and its c-invariants, the strategy constructs a path set whose actions allow to form a shortest solution to the problem. Consequently, the remaining actions are irrelevant and their removal yields a reduced planing problem.

In the beginning, we present a simple technique which we then refine in several steps. Each refinement consists of the identification of a class of paths that are irrelevant and the exclusion of these paths improves the strength and computational efficiency of the construction technique. In the next chapter we discuss an algorithm that finds the path set of the final strategy.

5.1 Outline

From the last chapter, we know that many paths are unnecessary for solving a planning problem because we can always use their replacement instead. But which paths are necessary? Surprisingly, the answer to this question often is: none. Many planning problems can be solved in several distinct ways such that we can always find a solution that does not use a given path. Then the natural next question is about a set of paths that allow to form a single solution. But we have already seen in Section 3.1.8 on Page 34 that finding even one path in this set is as hard as planning itself. So how can we hope to construct a set of necessary paths?

What we can do is to search for a superset of those paths that are necessary to form a solution. More precisely, we aim at an efficient way to construct a small set of paths that is complete. Hereby, the completeness of a path set means that for a solvable planning problem the actions of this path set suffice to form a shortest solution. The topic of this chapter is the development and analysis of a technique to find such sets; we call this technique path reduction.

The idea behind path reduction is simple: Extend an empty path set by paths that appear to be important while at the same time remove those paths that are identified as unnecessary given the set of paths accumulated so far. Terminate this process as soon as it converges to a fixed point, i. e., a path set that includes all necessary paths and from which all unnecessary have been removed. Finally, construct the reduced planning problem from the actions that are used by the paths in the fixed point together with

necessary actions of the unstructured part of the planning problem. Note that the result of path reduction is not the set of paths but a new planning problem, such that it can be the input of today's planning systems. The design of a planning system that takes as input the paths of a fixed point is left as further work.

The remainder of this chapter is organized as follows: First, we introduce a data structure, called path structure, that resembles the path set which is under investigation at a certain time. Furthermore, we give a simple strategy to extend path structures. We show that this first technique has advantageous properties, e. g., yielding a unique fixed point which preserves a shortest solution. Then we introduce several classes of paths that are unnecessary for shortest solutions and exempt each such class by creating a refined path structure and its associated way of extension. We show that each refinement preserves the properties of the previous one. The final strategy yields fixed points with all the desired properties. Now let us begin by defining the basic notion of path structure.

5.2 Path Structures

The goal of path reduction is to construct a set of paths which suffices to form a solution. The basic idea is to start with an empty set and to add paths until we reach a fixed point. But which paths are we supposed to add? For sure, we should add a path through a c-invariant that connects the fact of that c-invariant in the initial state to the one in the goal. To identify further paths, the data structure consists also of a set of potential beginnings and a set of potential endings of paths. If every path in the path set connects such a beginning to such an ending then we call the tuple a path structure.

Definition 5.1 (Type-0 Path Structures)
A path structure *of a planning problem $\Psi = (\mathcal{O}, \mathcal{I}, \mathcal{G})$ is a triple $(\mathcal{P}, \mathcal{B}, \mathcal{E})$ of a set $\mathcal{P}$ of paths of Ψ and two subsets $\mathcal{B}$, $\mathcal{E}$ of the union of all c-invariants $\mathcal{F}_\Psi^\mathcal{C}$ for which the following holds:*

- *$\mathcal{B}$ subsumes $\mathcal{I}$ and $\mathcal{F}_\Psi^{through}(\mathcal{P})$.*
- *$\mathcal{E}$ subsumes $\mathcal{G}$ and $\mathcal{F}_\Psi^{through}(\mathcal{P})$.*
- *Every path P in $\mathcal{P}$ begins with a fact in $\mathcal{B}$ and ends with a fact in $\mathcal{E}$.*

Also, we refer to this kind of path structure as type-0 path structure. Furthermore, we call $(\emptyset, \mathcal{I} \cap \mathcal{F}_\Psi^\mathcal{C}, \mathcal{G}')$ the empty *type-0 path structure for planning problem Ψ, where $\mathcal{G}'$ is the union of $\mathcal{G} \cap \mathcal{F}_\Psi^\mathcal{C}$ and $\mathcal{F}_\Psi^{through}(\emptyset)$.* ◇

In the following, we will define several refinements of this type of path structure. Because it is the basic type, we call it type-0.

The role of the sets $\mathcal{B}$ and $\mathcal{E}$ is as follows: The paths of a path structure *b*egin with facts in set $\mathcal{B}$ and *e*nd with the facts in $\mathcal{E}$, hence their name. We will see that most of the facts in $\mathcal{B}$ and $\mathcal{E}$ are determined by the sidepaths of paths in $\mathcal{P}$. The following notation allows us to refer to the facts in $\mathcal{B}$ and $\mathcal{E}$ in a convenient way.

Definition 5.2 (Starts and Stops of a Path Structure)
If $\Pi = \left(\mathcal{P}, \mathcal{B}, \mathcal{E}\right)$ is a path structure then we call a fact in $\mathcal{B}$ a start *of Π and a fact in $\mathcal{E}$* a stop *of Π.* ◇

The function $\mathcal{F}^{through}$ has been defined in Section 3.12 on Page 33. Applied on the actions of a path, it yields the facts of $\mathcal{F}^{\mathcal{C}}$ on which the path might depend. A path structure regards the dependencies of paths on the unstructured part by adding these facts to its sets of starts and stops. The goal of the empty type-0 path structure is extended by $\mathcal{F}^{through}(\emptyset)$ to account for the stops that result from facts of $\mathcal{F}^{\overline{\mathcal{C}}}$ in the goal.

Our idea of path reduction is to add interesting paths to a path structure until we cannot find a path that has not already considered. The following defines a corresponding terminology.

Definition 5.3 (Paths In a Path Structure and Regarding a Path Structure)
Let $\Pi = \left(\mathcal{P}, \mathcal{B}, \mathcal{E}\right)$ be a path structure and P be a path. Then we say that P is in *Π if $P \in \mathcal{P}$. We say that P is a path* regarding *Π if P begins with a start of Π and ends with a stop of Π.* ◇

We define the *extension* of a path structure for each different type separately. Generally said, a path can extend a path structure of a certain type if it is a path regarding that path structure, is not already in the path structure, and satisfies certain properties associated with that type. For type-0 path structures there are no such additional properties.

Definition 5.4 (Extension and Completeness of a Path Structure)
A path P can extend *a type-0 path structure Π if it is a path regarding Π and it is not in Π. A path structure of any type is* complete *if there is no path that can extend it according to its type.* ◇

Example 5.2 on Page 69 demonstrates the use of the terminology given in the preceding definitions. Note that complete type-0 path structures are typically infinite because their paths have circles.

Our goal is to find path sets whose actions allow to form a solution. For such path structures we say that they cover a solution.

Definition 5.5 (The Coverage of Paths)
We say that a path set $\mathcal{P}$ covers *a plan T if for every path-action o in T there is a path P in T such that o is in P and P is in $\mathcal{P}$. We say that a path structure $\left(\mathcal{P}, \mathcal{B}, \mathcal{E}\right)$* covers *a plan T if $\mathcal{P}$ covers T.* ◇

The result of path reduction should allow to find a solution to the original problem, if that problem was solvable in the first place. This property is sometimes called completeness. Note that there are other notions of completeness, e. g., to guarantee all solutions or to guarantee a single but arbitrary solution. Because we only guarantee completeness in the first sense, and because we use the term complete with a different meaning, we call this property *sufficiency*.

In the general case, the actions of a path set that covers a solution does not suffice to form that solution: Some actions of the solution are in $\mathcal{O}^{\overline{\mathcal{C}}}$. We have to consider actions of $\mathcal{O}^{\overline{\mathcal{C}}}$ in two respects: On the one hand, they might require paths that were not necessary otherwise. Path structures regard this point by including the facts of $\mathcal{F}^{through}$ in their

starts and stops. Furthermore, we have to guarantee that all necessary actions in $\mathcal{O}^{\overline{c}}$ are preserved, i. e., that they are part of the reduced planning problem. The function $\mathcal{O}^{nec}$, which is defined in Section 3.1.7 on Page 31, allows us to find the necessary actions of $\mathcal{O}^{\overline{c}}$ for a covering path set.

Definition 5.6 (Sufficiency of Path Structures)
A path structure Π suffices *for a planning problem Ψ if either (1) there is a shortest solution S to Ψ whose actions are all elements of $\mathcal{O} \cup \mathcal{O}^{nec}_{\Psi}(\mathcal{O})$, where $\mathcal{O}$ is the set of actions that are in Π, or (2) Ψ is not solvable.* ◇

Before we now show that coverage implies sufficiency, let us first examine the reason why a specific action is in a shortest solution.

Lemma 5.1 (About Actions in Shortest Solutions)
Let S be a shortest solution to a planning problem $(\mathcal{O}, \mathcal{I}, \mathcal{G})$. Then every action o in S adds a fact f that is false in the state yielded by executing the prefix of S before o in $\mathcal{I}$ and either (1) an action o' after o has f as precondition and o is the last action before o' that adds f, or (2) f is in $\mathcal{G}$ and action o is the last action that adds f. □

Proof: The removal of o from S yields a sequence that either has a conflict or whose execution in the initial state does not yield a state that entails the goal. Therefore, o adds a fact f that was previously false and that is either in the preconditions of a later action o' or in the goal. Furthermore, o is the last action before action o' or the goal, respectively, that adds f. (q. e. d.)

The preceding lemma allows us to show that a covering path structure is sufficient for a shortest solution to a planning problem.

Theorem 5.2 (Sufficiency of Covering Path Structures)
If a path structure covers a shortest solution to a planning problem then it suffices for that problem. □

Proof: Let Ψ be a planning problem, S a shortest solution to Ψ, Π a path structure of Ψ that covers S, and $\mathcal{O}$ the set of actions that are in Π. Then all path-actions in S are in $\mathcal{O}$ by definition. What about the actions in S that are in $\mathcal{O}^{\overline{c}}_{\Psi}$?

Let o be an action in $\mathcal{O}^{\overline{c}}_{\Psi}$. As o cannot be removed from S, we know from Lemma 5.1 that it adds a fact f in $\mathcal{F}^{\overline{c}}_{\Psi}$ that is either in $\mathcal{G}$ or that is in the precondition of an action o' after o in S. If f is in $\mathcal{G}$ then o is in $\mathcal{O}^{nec}_{\Psi}(\mathcal{O})$ by definition. If f is a precondition of an action o' then we know the following: If o' is a path-action then o is in $\mathcal{O}^{nec}_{\Psi}(\mathcal{O})$. If otherwise o' is in $\mathcal{O}^{nec}_{\Psi}(\mathcal{O})$ then o is an action in $\mathcal{O}^{nec}_{\Psi}(\mathcal{O})$, too. Therefore, the lemma holds if o is the last action of $\mathcal{O}^{\overline{c}}_{\Psi}$ in S. By going backwards, we observe that all actions in $\mathcal{O}^{\overline{c}}_{\Psi}$ are in $\mathcal{O}^{nec}_{\Psi}(\mathcal{O})$. (q. e. d.)

In other words, we can search for a complete path structure by considering path-actions alone. If the complete path structure covers all path-actions of a solution then it allows to find the necessary actions in $\mathcal{O}^{\overline{c}}$, too.

Now let us stipulate the following conventions for reasons of simplicity: If we consider a single and fixed planning problem then we sometimes omit to reference the problem. Likewise, if we refer to a single and fixed path structure then we sometimes omit to reference it, too.

5.3 Unnecessary Paths

The example in the previous section shows that complete type-0 path structures are too large to be useful, hence we have to find smaller ones. It is easy to find paths that might be necessary for a solution but it is hard to decide whether a path can be left out. Even if we are sure about rejecting single paths, how about their combined removal? We will see that each step in the construction of complete path structures depends on the accepted and rejected paths of all preceding steps, so we have to carefully study the consequences of rejections. Before we go into details, we give a short overview of these classes of paths that can be exempt.

5.3.1 Paths Longer than $2^{|\mathcal{F}_\Psi|}$

A shortest solution to planning problem Ψ is always shorter than $2^{|\mathcal{F}_\Psi|}$, thus there is no need to consider paths longer than this bound.

5.3.2 Path Composition

Another class of unnecessary paths are those that are a composition of others. For example, the blocks world path $P = \big(\langle \textit{move-onto}(C, A), \textit{move-from}(C, B), \textit{move}(C, B, D)\rangle, C^l_C\big)$ is equivalent to the sequence of the path $\big(\langle \textit{move-onto}(C, A)\rangle, C^l_C\big)$ and the path $P_2 = \big(\langle \textit{move-from}(C, B), \textit{move}(C, B, D)\rangle, C^l_C\big)$. In addition, we allow overlapped composition. For example, we consider path P to be a composition of the paths $\big(\langle \textit{move-onto}(C, A), \textit{move-from}(C, B)\rangle, C^l_C\big)$ and P_2.

5.3.3 Replaceable Paths

Intuitively, the class of replaceable paths is unnecessary. From our examination of path replacement we know that if a path is free in a solution then its replacement yields another solution. Unfortunately, there are two problems with the exclusion of replaceable paths: First of all, we have to guarantee that the rejected paths are indeed free. If a replaceable path is necessary for all solutions to a problem and it is always bound then its rejection yields insufficient path sets.

The second problem results from disappearing sidepaths. The beginnings and endings of sidepaths are important for path reduction because they are considered the endings and beginnings of additional paths, respectively. A replacing path can have fewer sidepaths than the one it replaces, thus the disregard of replaceable paths can eliminate paths from the final path set. We see this effect in the following example.

Example 5.1 (Disappearing Sidepaths)
Consider a planning problem of the blocks world that is commonly known as the Sussman anomaly[1] *(cf. Example 3.16 on Page 36 for the definition of the blocks world). It consists of three blocks A, B, and C which are configured according to the initial state $\mathcal{I} = \{clear(C), on(C, A), on\text{-}table(A), clear(B), on\text{-}table(B)\}$. The goal is to place block A onto B and block B onto C, i. e., $\mathcal{G} = \{on(A, B), on(B, C)\}$.*

Suppose we have two additional blocks D and E, where the first sits on the table and the second sits on the first. Our knowledge of the blocks world tells us that there is no need to consider these blocks for solving the goal above. In particular, there is no need to place block A onto D. This fact is reflected by path replacement, as every move over block D, can be replaced by a corresponding move over the table, e. g., $P_1 = (\langle move(A, B, D), move(A, D, C)\rangle, C^l_A)$ is replaceable by $P_2 = (\langle move\text{-}onto(A, B), move\text{-}from(A, C)\rangle, C^l_A)$.

Paths P_1 and P_2 are not truly equivalent. In contrast to the first, the second path does not have a sidepath through C^c_D, thus after rejecting replaceable paths, the fact $clear(D)$ is not important anymore. ■

The case of disappearing sidepaths is not always as easy as in this example and a major part of this chapter is dedicated to show that the use of path replacement yields sufficient path structures.

5.3.4 Arbitrary Goals

Planning problems usually do not provide a complete goal description, they just specify a subset of a possible goal state. Consequently, some c-invariants do not have a fact that is mentioned by the goal. For example, the standard formalization of the Sussman anomaly, as given in Example 5.1 on the page before, leaves the final position of block C unspecified, i. e., $C^c_C \cap \mathcal{G} = \emptyset$. We have to guarantee that path reduction still finds all paths in C^c_C that are necessary for a shortest solution, although we might never know that solution, i. e., never have explicit knowledge of the active fact of C^c_C after its execution.

Simple versions of path structures require complete goal descriptions. If the goal is not complete then they enrich the stop set of their empty path structure by all facts of c-invariants that are not mentioned in the goal. For example, the empty path structure of the Sussman anomaly would then include all locations of block C as stop. The final version of path reduction accepts arbitrary goals, i. e., leaves the goal unchanged.

5.4 Unrestricted Paths of Finite Length

Let us start with an easy case: Our first strategy shall accept paths up to a upper bound, i. e., we do not consider path replacement. Furthermore, we enrich the set of stops by all facts of c-invariants that are not mentioned by the goal. Then we are sure not to miss any important path and it is easy to show that the strategy finds a unique and sufficient fixed point. We say that path structures with these properties are of *type-1*.

[1] Although this problem has been named after G.J. Sussman, it was first discussed by Allen Brown.

Definition 5.7 (Type-1 Path Structures)
A path structure $\Pi = \big(\mathcal{P}, \mathcal{B}, \mathcal{E}\big)$ *of a planning problem* $\Psi = \big(\mathcal{O}, \mathcal{I}, \mathcal{G}\big)$ *is of* type-1 *if either one of the following two cases holds:*

1. $\big(\mathcal{P}, \mathcal{B}, \mathcal{E}'\big)$ *is the empty type-0 path structure for* Ψ*,* $\mathcal{E} = \mathcal{E}' \cup \mathcal{H}$*, and* $\mathcal{H}$ *is the set* $\bigcup_{C \in \mathcal{C}_\Psi, C \cap \mathcal{G} = \emptyset} C$*. In this case, we call* Π *the* empty type-1 path structure *for* Ψ*.*
2. $\Pi' = \big(\mathcal{P} \setminus \{P\}, \mathcal{B}', \mathcal{E}'\big)$ *is a type-1 path structure, where* P *is a path that can extend* Ψ'*. Furthermore,* $\mathcal{B}$ *is the union of* $\mathcal{B}'$*, all regular endings of (proper) sidepaths of* P*, and* $\mathcal{F}_\Psi^{through}(P)$*. Likewise,* $\mathcal{E}$ *is the union of* $\mathcal{E}'$*, all beginnings of (proper) sidepaths of* P *and* $\mathcal{F}_\Psi^{through}(P)$*.*

A path can extend *a type-1 path structure* Π *if it can extend the type-0 path structure,* P *is not in* Π*, and the length of* P *is* $2^{|\mathcal{F}_\Psi|} - 1$ *or shorter.* ◇

In the definition of type-1 path structures, the set $\mathcal{H}$ comprises the union of all c-invariants which do not have a fact in the goal state. By adding the facts of $\mathcal{H}$ to the set of stops, we assure that for every path-action of C in a solution there is a path P through C in Π that uses this action. We say that such path structures have a *complete stop set*.

Definition 5.8 (Path Structures with a Complete Stop Set)
A path structure $\Pi = \big(\mathcal{P}, \mathcal{B}, \mathcal{E}\big)$ *of a planning problem* $\big(\mathcal{O}, \mathcal{I}, \mathcal{G}\big)$ *has a* complete stop set *if for every c-invariant* C *of* Ψ *with* $\mathcal{G} \cap C = \emptyset$ *the set* C *is a subset of* $\mathcal{E}$*.* ◇

We have seen a similar kind of complete stop set before: In Example 3.2 on Page 17, which explains the analogy between a c-invariant and a transition system, we use the set C as the set of goal states in case the goal of a planning problem does not mention a fact of C. The drawback of complete stop sets is that type-1 path structures may have many unnecessary stops, thus are much larger than necessary. In Section 5.7 on Page 85, we will see that there is no need to extend the stop set by the set $\mathcal{H}$.

Example 5.2 (Type-1 Path Structures)
Reconsider the Sussman anomaly of Example 5.1 on Page 67. The empty type-1 path structure of this problem is $\Pi_0 = \big(\emptyset, \mathcal{I}, \mathcal{G} \cup \mathcal{H}\big)$*, where* $\mathcal{H}$ *comprises all facts of c-invariants but those in* C_A^l*,* C_B^l*,* C_B^c*, and* C_C^c*.*

$P = \big(\langle \mathit{move\text{-}onto}(C, A)\rangle, C_C^l\big)$ *is a path regarding* Π_0 *but is not in* Π_0*, thus it can extend* Π_0 *and the extension yields* $\Pi_1 = \big(\{P\}, \mathcal{I} \cup \{\mathit{clear}(A)\}, \mathcal{G} \cup \{\mathit{clear}(A)\}\big)$*. Therefore, fact* $\mathit{clear}(A)$ *is both a start and a stop. Note that the complete type-1 path structure of this problem is still too large to be useful: A state of the Sussman anomaly consists of* 12 *facts, so the fixed point comprises paths with lengths up to length* $2^{12} - 1$*.* ■

Let us now show the properties of type-1 path structures. First of all, it is easy to see that the complete one is unique.

Theorem 5.3 (Uniqueness of the Complete Type-1 Path Structure)
The complete type-1 path structure is unique for every planning problem. □

Proof: Extending a path structure is confluent, i. e., if two paths P_1 and P_2 can extend a type-1 path structure then the order of extension does not matter. As there is a unique smallest type-1 path structure, i. e., the empty path structure, there is also a unique largest one. (q. e. d.)

Likewise, it is easy to see that a complete type-1 path structure suffices to form a shortest solution to the underlying planning problem.

Theorem 5.4 (Complete Type-1 Path Structures are Sufficient)
The complete type-1 path structure of a planning problem Ψ suffices for Ψ. □

Proof: The size of a state is given by the number of facts $|\mathcal{F}_\Psi|$. Consequently, the number of distinct states is $2^{|\mathcal{F}_\Psi|}$ and every plan of length $2^{|\mathcal{F}_\Psi|}$ or larger must visit some state twice. Such loops can be removed, hence a shortest solution to Ψ has a length of less than $2^{|\mathcal{F}_\Psi|}$.

If planning problem Ψ has no solution then Π suffices by definition. Otherwise, let S be a shortest solution to Ψ. Path structure Π contains all paths of length $2^{|\mathcal{F}_\Psi|}$ and shorter. In particular, Π contains all paths in S that begin with a fact in $\mathcal{I}$ and end with a fact in $\mathcal{G}$, thus it covers S. Then we know from Theorem 5.2 on Page 66 that Π suffices for Ψ. (q. e. d.)

Note that a complete type-1 path structure covers every solution to a planning problem. It does not necessarily cover every plan of that problem. For this reason, we will always refer to solutions instead to arbitrary plans.

5.5 Minimal Paths

Obviously, type-1 path structures are too big to be of practical use. Fortunately, we can neglect many paths, e. g., paths that are a composition of others. We now show that it suffices to consider path structures that consist solely of uncomposed paths. Before we define a corresponding terminology, let us stipulate some conventions.

Definition 5.9 (Starts and Stops of c-Invariants)
Let $(\mathcal{P}, \mathcal{B}, \mathcal{E})$ be a path structure and C a c-invariant. Then a start *of C is a fact $f \in C$ that is in $\mathcal{B}$. Likewise, a* stop *of C is a fact in $\mathcal{E} \cap C$.* ◇

In the motivation, we said that a path is composed if it is overlapped by other paths. As paths in a path structure begin and end with a start and a stop, respectively, the fact that a path is composed depends on the starts and stops that it passes through. More precisely, the path has to pass through these facts in a certain order, i. e., pass through a start and then through a stop either simultaneously with that start or later. The active facts during the execution of a plan are determined by the pre- and postconditions of the action of the plan. We refer to the active facts and their order by the following terminology.

Definition 5.10 (Facts and Their Order in Sequences)
We say that a fact f is in *a sequence T if an action o in T has f as precondition or postcondition. Furthermore, we say that f is* before (after) *an action o' in T if o' is an action in T and f is in the prefix of T before o' (in the postfix of T after o'). Let f and f' be two facts in T. We say that f is* before (after) *f' in sequence T if there is an action o in T, f is* before (after) *o in T, and f' is after (before) o in T. Facts f and f' are simultaneously in T if o has both facts f and f' as precondition or as postcondition.* ◇

We also use the terminology of the previous definition for starts and stops of c-invariants in sequences. Now we elaborate on minimal paths, i. e., paths that are not a composition of others.

Definition 5.11 (Minimal Paths)
We say a path P is composed *regarding a path structure Π if a proper infix of P passes through a start and a stop of Π and the start is either before or simultaneously with the stop. A path is* minimal *if it is not composed.* ◇

Consequently, paths of length one are minimal regarding every path structure. Note that a path that is not a path regarding a path structure Π can still be minimal or composed regarding Π. Remember that we omit to reference path structures if we refer to a single and fixed one.

Example 5.3 (Minimal Paths)
Let us consider a path structure $\Pi = \big(\mathcal{P}, \{f_1, f_3\}, \{f_2, f_4\}\big)$, a sequence $T = \langle o_1, o_2, o_3\rangle$, and two paths $\big(T, C_1\big)$ and $\big(T, C_2\big)$. Furthermore, let $pre(o_2) \cap C_1 = \{f_1\}$ and $post(o_2) \cap C_1 = \{f_2\}$. Then fact f_1 is a start of C_1 in T and fact f_2 is a stop of C_1 in T. Therefore, a start of C_1 is before a stop of C_1 in T and (T, C_1) is composed. If $pre(o_3) \cap C_2 = \{f_4\}$ is the only start and $post(o_1) \cap C_2 = \{f_3\}$ is the only stop of C_2 in T then there is neither a start of C_2 before a stop of C_2 in T nor are a start of C_2 and a stop of C_2 simultaneously in T. Therefore, $\big(T, C_2\big)$ is not composed, thus it is minimal. ■

If a path is the composition of a set of paths then the sequence of the former is covered by the sequences of the latter. Such covered paths do not contribute to the set of starts and stops of a path structure.

Lemma 5.5 (About Starts and Stops of a Composed Path)
Let P be a path regarding a path structure Π. If P is composed and covered by Π then every beginning of a sidepath of P is a beginning of a sidepath of a path in Π. Likewise, every ending of a sidepath of P is an ending of a sidepath of a path in Π. Furthermore, if f is a fact in $\mathcal{F}_\Psi^{through}(P)$ then there is a path P' in Π and $f \in \mathcal{F}_\Psi^{through}(P')$. □

Proof: There is a subset $\mathcal{P}$ of paths of Π such that every path in $\mathcal{P}$ is a subsequence of P and for every action o in P there is a path P' in $\mathcal{P}$ such that o is in P'. Thus, if $f \in \mathcal{F}_\Psi^{through}(o)$ then $f \in \mathcal{F}_\Psi^{through}(P')$. Furthermore, if o is the first action of a sidepath through C' of P then there is a path P' in $\mathcal{P}$ such that o is the first action of a sidepath through C' of P'. The same is true if o is the last action of a sidepath of P. (q. e. d.)

Now, we know that we can remove composed paths from path structures.

Definition 5.12 (Type-2 Path Structures)
A path structure $\Pi = \left(\mathcal{P}, \mathcal{B}, \mathcal{E}\right)$ of a planning problem Ψ is of type-2 *if at least one of the following three cases holds:*

1. *Π is the empty type-1 path structure, i. e. the path structure $\left(\emptyset, \mathcal{I} \cap \mathcal{F}_\Psi^\mathcal{C}, \mathcal{G}'\right)$, where $\mathcal{G}'$ is the union of $\mathcal{G} \cap \mathcal{F}_\Psi^\mathcal{C}$ and $\mathcal{F}_\Psi^{through}(\emptyset)$.*
2. *$\Pi' = \left(\mathcal{P} \setminus \{P\}, \mathcal{B}', \mathcal{E}'\right)$ is a type-2 path structure, where P is a path that can extend Π'. Furthermore, $\mathcal{B}$ is the union of $\mathcal{B}'$, all regular endings of (proper) sidepaths of P, and $\mathcal{F}_\Psi^{through}(P)$. Likewise, $\mathcal{E}$ is the union of $\mathcal{E}'$, all beginnings of (proper) sidepaths of P, and $\mathcal{F}_\Psi^{through}(P)$.*
3. *$\Pi' = \left(\mathcal{P} \cup \{P\}, \mathcal{B}, \mathcal{E}\right)$ is a type-2 path structure, where $P \notin \mathcal{P}$ and P is composed regarding Π'.*

Hereby, a path can extend a type-2 path structure Π if it can extend the type-1 path structure Π and it is minimal regarding Π. ◇

Point 3 of the preceding definition reduces the number of paths of type-2 path structures in comparison to type-1 ones. Example 5.4 on the next page demonstrates the process of constructing a complete type-2 path structure.

Type-2 path structures have the same advantageous properties as those of type-1.

Theorem 5.6 (Uniqueness of the Complete Type-2 Path Structure)
For every planning problem the complete type-2 path structure is unique. □

> **Proof:** The complete type-1 path structure is unique (cf. Theorem 5.3 on Page 69). Furthermore, we know from Lemma 5.5 on the preceding page that a path is minimal in respect to a complete type-2 path structure Π if and only if it is minimal in respect to the complete type-1 path structure. In other words, despite the removal and prevention of composed paths, a complete type-2 path structure has the same starts and stops as the complete type-1 path structure. As each complete type-2 path structure comprises all minimal paths of the complete type-1 path structure, and it does not contain additional paths, there is actually only one such complete type-2 path structure. (q. e. d.)

Theorem 5.7 (The Complete Type-2 Path Structure Suffices)
The complete type-2 path structure for a planning problem Ψ suffices for Ψ. □

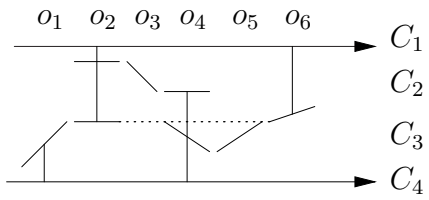
> **Proof:** For each planning problem, the complete type-2 path structure Π_2 covers the same plans that are covered by the complete type-1 path structure Π_1: If a path P is in Π_1 but not in Π_2 then it is composed regarding Π_2. In other words, P is covered by Π_2. Therefore, the sufficiency of the complete type-2 path structure follows from the sufficiency of the complete type-1 path structure. (q. e. d.)

Now we are ready to exclude replaceable paths.

Example 5.4 (Construction of a Type-2 Path Structure)
Let us demonstrate how to find a complete type-2 path structure for the planning problem $\Psi = \big(\{o_1, \ldots, o_6\}, \{f_1, f_4, f_6, f_{10}\}, \{f_3, f_5, f_{12}\}\big)$. *The actions and c-invariants of* Ψ *are given by the table*

c-invariants	o_1	o_2	o_3	o_4	o_5	o_6
$C_1 = \{f_1, f_2, f_3\}$		$f_1 \to f_2$				$f_2 \to f_3$
$C_2 = \{f_4, f_5\}$		f_4	$f_4 \to f_5$	f_5		
$C_3 = \{f_6, f_7, f_8, f_9\}$	$f_6 \to f_7$	f_7		$f_7 \to f_8$	$f_8 \to f_7$	$f_7 \to f_9$
$C_4 = \{f_{10}, f_{11}, f_{12}\}$	$f_{10} \to f_{11}$			$f_{11} \to f_{12}$		

and the following graph depicts the only solution $\langle o_1, \ldots, o_6\rangle$ *to this problem.*



We now construct a complete type-2 path structure for this planning problem. Hereby, we report the path structures $\Pi_i = \big(\mathcal{P}_i, \mathcal{B}_i, \mathcal{E}_i\big)$, *where* Π_0 *is the empty type-2 path structure and* Π_{i+1} *is the path structure after connecting all starts and stops present in path structure* Π_i *and only those, excluding all paths that are composed. We denote paths with* $P^i_{p,q} = \big(T^i_{p,q}, C_p\big)$, *where* i *refers to the index of path structure,* p *is the index of the c-invariant, and* q *is the index of the path.*

1. $\Pi_0 = \big(\emptyset, \mathcal{I}, \mathcal{G} \cup \mathcal{H}\big)$ *is the empty type-2 path structure. The set* $\mathcal{H}$ *comprises the facts of* C_3*, the only c-invariants that is not mentioned in the goal.*

2. *The first iteration adds all paths that go from the initial state to* $\mathcal{G} \cup \mathcal{H}$. *This yields the path structure* Π_1 *with* $\mathcal{P}_1 = \{P^1_{1,1}, P^1_{2,1}, P^1_{3,1}, P^1_{3,2}, P^1_{3,3}, P^1_{3,4}, P^1_{3,5}, P^1_{4,1}\}$, *where* $T^1_{1,1} = \langle o_2, o_6\rangle$, $T^1_{2,1} = \langle o_3\rangle$, $T^1_{3,1} = \langle o_1\rangle$, $T^1_{3,2} = \langle o_1, o_4\rangle$, $T^1_{3,3} = \langle o_1, o_4, o_5\rangle$, $T^1_{3,4} = \langle o_1, o_6\rangle$, $T^1_{3,5} = \langle o_1, o_4, o_5, o_6\rangle$, *and* $T^1_{4,1} = \langle o_1, o_4\rangle$. *Consequently,* $\mathcal{B}_1 = \mathcal{B}_0 \cup \{f_3, f_5, f_7, f_8, f_9, f_{11}, f_{12}\}$ *and* $\mathcal{E}_1 = \mathcal{E}_0 \cup \{f_2, f_4, f_5, f_{10}\}$.

3. *Now, we add the paths with sequences* $T^2_{1,1} = \langle o_2\rangle$, $T^2_{3,1} = \langle o_4\rangle$, $T^2_{3,2} = \langle o_5\rangle$, *and* $T^2_{4,1} = \langle o_4\rangle$. *Therefore,* $\mathcal{B}_2 = \mathcal{B}_1$ *and* $\mathcal{E}_2 = \mathcal{E}_1 \cup \{f_7, f_{11}\}$. *We disregard the composed paths* $P^1_{3,2}$, $P^1_{3,3}$, $P^1_{3,4}$, $P^1_{3,5}$, *and* $P^1_{4,1}$.

4. *Finally, we extend* Π_3 *by the path* $P^3_{4,1} = \big(\langle o_1\rangle, C_4\big)$, *which yields* $\mathcal{B}_4 = \mathcal{B}_3$ *and* $\mathcal{E}_4 = \mathcal{E}_3$. *With* Π_4, *the process reaches a fixed point.*

All minimal paths regarding Π_4 *are free. In addition,* Π_4 *covers* S, *e. g., with the path set* $\big\{P^1_{1,1}, P^1_{2,1}, P^2_{3,2}, P^3_{4,1}, P^2_{4,1}\big\}$.
Clearly, the smallest type-2 path structure is too large to serve as useful reduction. In Example 5.6 on Page 88 we find a similar path structure for the same problem that does not use a complete stop set. The resulting smallest complete type-4 path structure for this problem comprises fewer paths. ■

5.6 Replaceable Paths

Intuitively, we know that the exclusion of replaceable paths improves a reduction technique based on path structures. Unfortunately, it is not immediately clear that their removal preserves sufficiency: For example, we can replace paths only if they are free. But are minimal paths in shortest solutions always free? Furthermore, a replacing path can have fewer sidepaths than the path that it replaces. As path minimality is related to the starts and stops of a path structure, fewer starts and stops result in more minimal paths. This circular dependency between the minimal paths of a solution and the starts and stops resulting from these paths prevent a simple proof that the exclusion of replaceable paths preserves sufficiency.

To overcome this problem, we examine path structures that have "enough" starts and stops; we call them *usable*. In particular, a usable path structure Π comprises all starts and stops that result from those paths of a shortest solution that are minimal regarding Π. Usability is weaker than coverage, because it makes no statement about specific actions, but it is strong enough to show that replaceable paths can be neglected. Furthermore, we will see that the complete type-2 path structure, i. e., the one from which we would prefer to exclude replaceable paths is usable. Therefore, the notion of usability allows to examine the exclusion of replaceable paths without running into the circular dependency.

Let us make a remark before we go into details: In the following, we mainly consider shortest solutions, i. e., we formulate theorems about shortest solutions to a planning problem. One reason is, of course, that shortest length is a desired optimality criterion and we prefer techniques that preserve optimality. The main reason, however, is that all parts of shortest solutions are necessary, i. e., there are no partial sequences of shortest solutions that can be removed. Of course, the reference to shortest solutions does not mean that the reduced problem has only solutions that are optimal solutions for the original problem. Nevertheless, we can assume that path reduction eliminates longer solutions more likely.

5.6.1 The Usability of Path Structures

Let us first state which paths we want to keep, i. e., which paths are relevant, and which paths we want to eliminate. In order to find relevant paths, we have to look at non-replaceability: A path can always replace itself and if it is replaceable by a different path, the second one might be replaceable by a third. Therefore, we introduce the term relevant for paths that cannot be replaced by another path. Then, our final path structure consists solely of relevant paths.

Definition 5.13 (Relevant Paths)
We call a path P relevant *if there is no path P' such that P' can replace P and $P' < P$. Hereby, $<$ is a lexicographic ordering on sequences with the length of the sequences as first criterion, i. e., $T_1 < T_2$ if T_1 is shorter than T_2, and an arbitrary but fixed total ordering on sequences of equal length as the second. A path is* irrelevant *if it is not relevant. We call a path P' a* relevant path for *P if it is relevant and can replace P.* ◇

The ordering relation $<$ prefers short paths. For example, if the second criterion orders action o_1 before action o_2 then we have $\big(\langle o_1\rangle, C\big) < \big(\langle o_2\rangle, C\big) < \big(\langle o_1, o_2\rangle, C\big)$. This way,

the relation $<$ guarantees that a shorter path is relevant, even if it is replaceable by a longer one. In other words, the replacement of a path by a relevant path in a plan always yields a plan of the same or shorter length. Note that there can be two or more relevant paths for a given path. In the following, the differentiation between these paths is of no further importance, i. e., in order to chose a relevant path for a given one, we just pick an arbitrary one.

As noted in the motivation on the previous page, the paths in a solution and those in a complete path structure mutually depend: Whether or not a path in a solution is a path regarding a path structure Π depends on the starts and stops of Π. In turn, the start and stops of Π depend on the paths that were used to extend that path structure. We break this cycle by identifying path structures with "enough" starts and stops, which allow to reason about the sidepaths of paths in the solution even if the paths are *not* in that path structure. We call such path structures usable.

Definition 5.14 (Usable Path Structures)
Let Ψ be a planning problem. We call a path structure Π usable *for a solution S of Ψ if for every minimal path P in S it holds that every beginning of a sidepath of P is a stop of Π, every ending of a sidepath of P is a start of Π, and every fact in $\mathcal{F}_{\Psi}^{through}(P)$ is both a start and a stop of Π. Furthermore, Π includes a relevant path for every path regarding Π that is a proper partial sequence of a minimal path in S.* ◇

The usability of a path structure for a solution guarantees that every beginning of a sidepath of a minimal path in that solution is a stop and every ending is a start. It does not say whether that path is in the path structure or not. The second criterion for a usable path structure Π is that if a minimal path P in a solution regarding Π has a circle then Π includes a relevant path P'' for P', where P' is the path P without that circle. This property is used by the upcoming proofs. Note that such paths are always comprised by a complete path structure.

5.6.2 About Minimal Paths

In this subsection we show lemmas about minimal paths. Minimal paths cannot have arbitrary sidepaths. In particular, a minimal path has either zero or one sidepath with core through a given c-invariant.

Lemma 5.8 (Lemmas about Minimal Paths)
Let Ψ be a planning problem, S a solution to Ψ, and Π a path structure with complete stop set that is usable for S. Then a path P_1 in S regarding Π is composed if at least one of the following is true:

1. *P_1 has a sidepath with core through C_2 and there is a start and a stop of C_2 in S between two adjacent path-actions o_1 and o_2 of C_2 in P_1.*
2. *P_1 has two or more sidepaths with core through the same c-invariant C_2.* □

Proof: Let P_1 be a path through c-invariant C_1. We show the first point as follows: Plan P_1 has a sidepath Q_1 through C_2. If Q_1 is the non-proper sidepath of P_1 then the lemma holds trivially. Otherwise, Q_1 is a proper

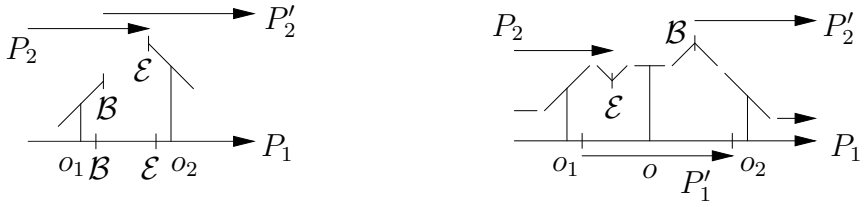


Figure 5.1: Graphs depicting configurations of paths and sidepaths that are used in two proofs: The left figure shows a configuration used in Lemma 5.8.2. The right figure shows a configuration of paths in Lemma 5.9. The labeling of a fact with $\mathcal{B}$ and $\mathcal{E}$ denotes that this fact is a start and stop of the path structure, respectively.

sidepath of P_1. Then the start and the stop of C_2 in S imply the existence of two paths through C_2 in S: Path P_2 ends before o_2 and has o_1 as last path-action of C_1. Path P_2' begins after o_1 and has o_2 as first path-action of C_1. Therefore, $post(o_1) \cap C_1$ is a start and $pre(o_2) \cap C_1$ is a stop. Furthermore, because o_1 and o_2 are adjacent path-actions of C_2 in P_1, the start is either before the stop in P or both are simultaneously, thus P_1 is composed.

As the ending of the first sidepath is a start of C_2 and the beginning of the second is a stop, the second assertion follows from the first. The configuration of paths and sidepaths in this case is depicted on the left side of Figure 5.1. (q. e. d.)

The next lemma discusses a special configuration of paths that we will use twice in the upcoming proofs: If an action o of a sidepath is during the core of this sidepath and the adjacent actions of the core enclose a start and a stop in a solution then the precondition of o is a start and a stop, too.

Lemma 5.9 (Lemma about Minimal Paths: A Special Case)
Let Ψ be a planning problem, S a shortest solution to Ψ, and Π a path structure of Ψ with a complete stop set that is usable for S. Let $P_1 = \big(\langle \ldots, o_1, \ldots, o, \ldots, o_2, \ldots \rangle, C_1\big)$ be a path in S regarding Π, where o is a proper sidepath-action of C_2, o_1 is the last path-action of C_2 before o in P_1, and o_2 is the first path-action of C_2 after o in P_1. If there is a start and a stop of C_2 in S between actions o_1 and o_2 then $pre(o) \cap C_2$ is a start and a stop. □

Proof: The start and the stop of C_2 in S imply the existence of two paths through C_2 in S: Path P_2 ends before o_2 and has o_1 as the last path-action of C_1. Path P_2' begins after o_1 and has o_2 as the first path-action of C_1. Therefore, $post(o_1) \cap C_1$ is a start and $pre(o_2) \cap C_1$ is a stop. Thus there is a path P_1' in S through C_1 that has the coreless sidepath $\big(\langle o \rangle, C_2\big)$. The configuration of paths and sidepaths in this case is depicted on the right side of Figure 5.1. (q. e. d.)

Besides minimal paths, we are also interested in shortest paths.

5.6.3 About Shortest Paths

In the following, we are often interested in a certain action and in minimal path through a given c-invariant with that action. But this path is not unique: Consider a minimal path $P = (\langle \ldots, o \rangle, C)$, which is a minimal path through C with action o. Furthermore, assume that a proper infix of P passes through a start. Then a proper postfix of P is a minimal path through C with action o, too. The notion of shortest paths allows us to reference such a path in a unique way.

Definition 5.15 (Shortest Paths)
Let Ψ be a planning problem, Π be a path structure of Ψ, T be a plan of Ψ, and o an action in T. Then the term path in T with action o *refers to any path P in T regarding Π with o as action. Furthermore, the term* shortest *path in T with action o refers to any path P in T with action o, such that there is no path P' regarding Π in T with action o and P' is shorter than P.* ◇

Note that although the shortest path with an action through a given c-invariant is unique, there can be another path of same length with o through a different c-invariant. In other words, a shortest path is unique only in respect to a given c-invariant. Let us give examples of shortest paths.

> **Example 5.5 (Shortest Paths)**
> *Let us consider shortest paths with action o_2 in the plan $T = \langle o_1, \ldots, o_5 \rangle$. Let us assume that there are exactly four paths $P_i = (T_i, C_i)$, $1 \leq i \leq 4$, in T that are paths regarding a path structure Π. Hereby, $T_1 = \langle o_2, o_5 \rangle$, $T_2 = \langle o_1, o_2 \rangle$, $T_3 = \langle o_1, o_2, o_3 \rangle$, and $T_4 = \langle o_3 \rangle$.*
>
> *Path P_1 is of length two and comprises action o_2. There is no shorter such path in T, thus P_1 is a shortest path with action o_2. P_2 is another shortest path in T with action o_2 and these are the only paths with this property: Although path P_3 uses action o_2 and lies during a shorter subsequence of T than P_1, its total length is three, which makes it longer than P_1. Path P_4 is shorter than path P_1 but it does not use action o_2.* ■

Shortest paths and minimal paths in a plan are related. In particular, only a minimal path can be a shortest path with a specific action. The following lemma formalizes this observation.

Lemma 5.10 (Shortest Paths with a Given Action are Minimal)
Let Ψ be a planning problem, S a solution to Ψ, and Π a path structure of Ψ. Let o be an action in plan S and P a shortest path in S regarding Π with action o. Then P is minimal regarding Π. □

> **Proof:** Let P be a path through c-invariant C and T the longest proper infix of P. Then there is no start of C in T before o and no stop of C in T after o. Therefore, P is minimal. (q. e. d.)

Note that this is true for arbitrary path structures, especially for those that do not have a complete stop set.

Figure 5.2: Graphs depicting paths in proofs of Lemma 5.11. The left figure shows the configuration of paths in Point 1 and the right figure in Point 3.

As with minimal paths, a shortest path with a given action cannot have arbitrary sidepaths. In particular, the action has to be during the core of every of its sidepaths.

Lemma 5.11 (Lemmas about Shortest Paths with a Given Action)
Let Ψ be a planning problem, S a solution to Ψ, and Π a path structure of Ψ with a compete stop set that is usable for S. Let o be an action in S and P_1 a shortest path in S regarding Π with action o. If P_1 has a sidepath Q_1 with core through C_2 then the following is true:

1. *If the core of Q_1 begins before o then there is no stop of C_2 in S between the first action of the core of Q_1 and action o.*

2. *If the core of Q_1 ends after o then there is no start of C_2 in S between o and the last action of the core of Q_1.*

3. *o is during the core of Q_1.* □

Proof:

1. Let us assume the lemma does not hold. Then there is a stop f of C_2 in S after the first action of the core of Q_1 and before action o. This configuration is depicted on the left side of Figure 5.2. Let o' be the last path-action of C_2 in P_1 that is before the stop f. Then there is a path P_2 through C_2 in S and o' is the last action of the core of a sidepath of P_2 through C_1. As the end of this sidepath is a start of C_1, P_1 is not the shortest path in S with action o.
2. Similar to the proof of Point 1.
3. The beginning of the core of Q_1 is a stop of C_2 and its ending is a start of C_2. Thus, this lemma is implied by Points 1 and 2. The right side of Figure 5.2 depicts a configuration in which P_1 is not a shortest path with action o because it is before the core. (q. e. d.)

We now continue with an examination of circular paths. All partial sequences of shortest solutions are necessary. Based on this observation, we can deduce several facts about circular paths.

5.6.4 About Circular Paths

Circular paths are interesting because of two reasons: On the one hand, if a sidepath is bound by a path-action then it is in fact bound by a circular path. On the other hand, the execution of a circular path has no effect on the active fact of its c-invariant. For this reason, any circular path without sidepaths and added effects in $\mathcal{F}^{\overline{C}}$ can be removed from solution plans. We now show that if a circular path is in a shortest solution, it passes through a start and a stop of its c-invariant, from which follows that no minimal path is bound by a circular path.

We begin by examining reasons why an arbitrary circular path is in a shortest solution. Hereby, arbitrary refers to the fact that these properties hold independently of a specific path structure. In Theorem 5.1 on Page 66 we have carried out a similar examination for actions in shortest solutions.

Lemma 5.12 (About Circular Paths in Shortest Solutions)
Let S be a shortest solution to a planning problem Ψ. Then for every circular path $P = (T, C)$ in S at least one of the following is true:

1. *The non-proper sidepath of path P is bound.*
2. *Path P has a proper sidepath that is non-circular.*
3. *Path P has a proper sidepath that is bound.*
4. *An action in path P adds a fact in $\mathcal{F}^{\overline{\mathcal{C}}}_{\Psi}$.* □

Proof: Let S' be the sequence S without sequence T. S is a shortest solution to Ψ, so either sequence S' has a conflict or its execution in the initial state yields a state that does not entail the goal. Both cases result from the removal of an action o that adds a fact f. More precisely, f is false before T, true after o, and is needed by a later action o' or by the goal.

We prove the lemma by case differentiation. Regarding fact f we differentiate between the cases (1) that f is a fact in c-invariant C, (2) that f is a fact in set $\mathcal{F}^{\mathcal{C}}_{\Psi}$ but not in c-invariant C, i. e., that f is in a c-invariant C' and $C' \neq C$, and (3) that f is a fact in set $\mathcal{F}^{\overline{\mathcal{C}}}_{\Psi}$. Regarding action o' we differentiate between the following cases: (a) There is no such action o' after action o in solution S, thus fact f is in the goal. Otherwise, (b) fact f is precondition of an action o'. Each case in the cross product of the two differentiations has either to be contradictory or to imply the lemma.

1. Fact f is in c-invariant C.
   (a) This case is inconsistent with the prerequisites: f is true before and after T, so the removal of T does not falsify f. Therefore, there has to be another reason why P cannot be removed.
   (b) Action o' lies during T, thus it binds the (non-proper) sidepath P (Point 1).
2. Fact f is in c-invariant C' with $C' \neq C$. Path (T, C) has a sidepath Q through c-invariant C'.

Figure 5.3: Graphs depicting the configuration of paths for the two cases of the proof of Lemma 5.13. On the left side, path P'_1 has a sidepath Q'_2 through C_2. On the right side, no such sidepath exists.

(a) Sidepath Q is non-circular (Point 2).

(b) Action o' is not in T. If it lies during sequence T then it binds sidepath Q (Point 3). Otherwise it is after sequence T and sidepath Q is non-circular (Point 2).

3. Fact f is in $\mathcal{F}_\Psi^{\overline{\mathcal{C}}}$. Then both cases for action o' imply Point 4.

These cases cover all possibilities. All valid cases imply the lemma and each of the four points given by the lemma is used by a valid case. (q. e. d.)

In other words, any circular path in a solution can be removed if it does not meet at least one of these four conditions.

In the following, we show that in all four cases a circular path passes through a start and a stop of its c-invariant. In other words, a path is composed if it is bound by a circular path. We begin by showing this property for two special cases. First, we cover the case where the circular path has a non-circular sidepath. In the second case, the non-proper sidepath of the circular path is bound. Finally, Lemma 5.15 on Page 82 shows that arbitrary circular paths pass through a start and a stop.

Lemma 5.13 (About Circular Paths with a Non-Circular Sidepath)
Let Ψ be a planning problem, S a shortest solution to Ψ, and Π a path structure of Ψ with a complete stop set that is usable for S. Let P be a circular path in solution S that is not necessarily a path regarding Π. If path P has a non-circular sidepath Q_2 then P passes through a start and a stop. □

Proof: First, we show that P passes through a stop. We then use the same argument to show that it passes through a start, too. Hereby, we use the following notations, which we depict in Figure 5.3.

- $P = (T, C_1)$ is a circular path in S that has a non-circular sidepath Q_2 through C_2. P_1 is the path of all path-actions of C_1 in S. P'_1 is the path P_1 without the sequence T and P''_1 is a relevant path for P'_1. Note that P''_1 is a path in Π.
- Q'_2 is the last sidepath through C_2 of P''_1 before the beginning of T, if such a sidepath exists. As T is non-circular in C, Q'_2 ends before T. Let f be the end of Q'_2. Therefore, f is a start of C_2.

- o' is the first path-action of C_2 in T. P_2 is the shortest path in S through C_2 with o'. Q_1 is the sidepath of P_2 through C_1 with o'.

We first show that the core of Q_1 begins during P: We know that Q_1 begins before the end of P because it includes o'. Regarding its earliest beginning in S, we differentiate two cases: Either (1) Q_2' exists or (2) no such sidepath exists. These cases are depicted on the left and right of Figure 5.3 on the preceding page, respectively.

In case (1) we know that f is a start of C_2. Therefore, P_2 begins after the end of Q_2' and the core of Q_1 begins simultaneously with P or later. In case (2) there is no path-action of C_1 in P_2 before P, thus the core of Q_1 begins simultaneously with P or later, and the assertion holds, too.

As the beginning of Q_1 is a stop of C_1, we have finished the first part. We use a similar argument to show that P passes through a start of C_1. This finalizes the proof. (q. e. d.)

The following lemma covers another special case of Lemma 5.15 on the following page: If the non-proper sidepath of a circular path is bound then the path passes through a start and a stop. Actually, it shows a stronger result: If the non-proper sidepath of a circular path is bound then the path is composed.

Lemma 5.14 (About Circular Paths that are Bound)
Let Ψ be a planning problem, S a shortest solution to Ψ, and Π a path structure of Ψ with a complete stop set that is usable for S. Let P be a circular path in S, not necessarily regarding Π. If the non-proper sidepath of P is bound then P is composed. □

Proof: We use the following notations which we depict in Figure 5.4 on the next page: P goes through c-invariant C_1, P_1 is a path in S that has P as subsequence, and o is an action that binds the non-proper sidepath of P. If o is in $\mathcal{O}_\Psi^{\overline{C}}$ then the lemma holds trivially. Otherwise, let o be path-action of c-invariant C_2 and P_2 be the shortest path through C_2 in S with action o. Q_1 is the sidepath of P_2 through C_1 with action o.

We show this lemma for all circular paths in S. Hereby, we consider these paths in the reverse order of their beginning in S. In other words, we start with those that begin the latest in S and work our way back towards the beginning of S. Hence, we can assume that all circular subsequences of P_1 and P_2, not necessarily regarding Π, that begin later than P pass through a start and a stop.

We differentiate between three cases regarding the core of Q_1: (1) Q_1 is coreless, (2) either o is before the beginning or after the ending of its core, and (3) o lies during the core of Q_1.

1. Q_1 is coreless. Then $pre(o) \cap C_1$ is a start and a stop, thus the lemma holds.

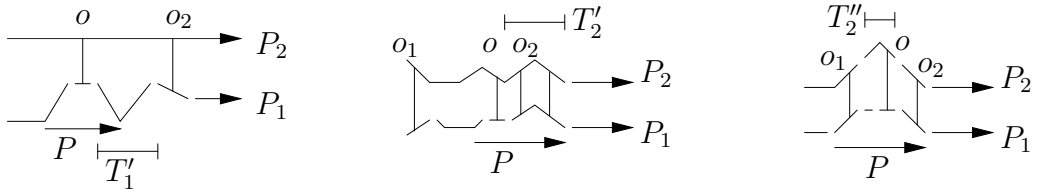


Figure 5.4: Graphs depicting configurations of paths that occur in the proof of Lemma 5.14.

Otherwise, Q_1 has a core. We now examine two path-actions of C_1 in Q_1: The last one before o and the first one after o. We call the first action o_1 and the second o_2.

2. Action o_1 exists and action o_2 does not, thus o lies in P_2 after the beginning of the core of Q_1. This case yields a contradiction: Lemma 5.11.2 on Page 78 states that $post(o_1) \cap C_2$ is a start, thus P_2 is not the shortest path in S through C_2 with action o. Point 1 of the same lemma implies the same if action o_2 exists but action o_1 does not.

3. Action o lies during the core of sidepath Q_2. Here, we consider three sub-cases, which all imply a circular path in P_1 or P_2 that begins later than P. We depict these cases in Figure 5.4:
   (a) If there are path-actions of C_1 between o and o_2 in S then o and o_2 enclose a circular subsequence T_1' of P_1. Sequence T_1' begins later than P in S, so according to our assumption it passes through a start and a stop of C_1. Then we know with Lemma 5.9 on Page 76 that $pre(o) \cap C_1$ is a start and a stop. We see this configuration in the left graph of Figure 5.4.
   Otherwise, there is no path-action of C_1 between o and o_2, thus o_2 lies during P. (b) If o_1 is before P then o_2 has to be part of a sidepath $\left(T_2', C_2\right)$ of P. The induced path of this sidepath begins later than P in S because there is no path-action of C_2 in P before o. If this sidepath is circular then it is bound. If it is not circular then we know with Lemma 5.13 on Page 80 that P_1 passes through a start and a stop of C_1. In both cases, P_2 is not a shortest path through C_2 with action o. This configuration is depicted in the middle graph of Figure 5.4.
   (c) If action o_1 is in P then we consider the subsequence T_2'' of P_2 between o_1 and o_2. T_2'' is not empty, it begins later than P in S, and it is circular. Then we know with Lemma 5.8.1 on Page 75 that P is composed. This configuration is depicted in the right graph of Figure 5.4.

The three cases imply that $pre(o) \cap C$ is a start and stop, thus P is composed. (q. e. d.)

The following lemma generalizes the results of the previous two: Arbitrary circular pats pass through a start and a stop.

Lemma 5.15 (Circular Paths Pass through a Start and a Stop)
Let Ψ be a planning problem, S a shortest solution to Ψ, and Π a path structure of Ψ with a complete stop set that is usable for S. Then every circular path P in S, not necessarily regarding Π, passes through a start and a stop. □

Proof: Lemma 5.12 on Page 79 gives four conditions, of which at least one must be true for every circular path in a shortest solution. For the first condition, the lemma follows from Lemma 5.14 on Page 81, for the second from Lemma 5.13 on Page 80, and in case the fourth condition applies then the lemma follows from Definition 5.1 on Page 64.

The only open case is Condition 3, where all sidepaths of P are circular and one of them, let us call it Q, is bound. In other words, the induced path P_2 of Q is bound. Then P_2 is composed, thus a proper infix of P_2 passes through a start and a stop. The start is a beginning of a sidepath through C with core and the first action of the core is in P. Therefore, P passes through a stop of C. With the same argument, we observe that a P passes through a start of C, thus the lemma holds. (q. e. d.)

The following lemma shows the validity of the first step towards paths structures that are free of irrelevant paths: Minimal paths regarding a usable path structure are free.

Theorem 5.16 (Minimal Paths are Free)
Let Ψ be a planning problem, S a shortest solution to Ψ, Π a path structure of Ψ with a complete stop set that is usable for S, and P a minimal path in S. Then P is free. □

Proof: We proof the lemma by contradiction. Let us assume that the path P has a sidepath Q through C and Q is bound by an action o. Firstly, if o is in $\mathcal{O}_\Psi^{\overline{C}}$ then Definition 5.1 on Page 64 states that fact $pre(o) \cap C$ is a start and a stop of Π, thus P is composed.

Otherwise, o is a sidepath-action of C. If o is a path-action of C then Q is bound by a circular path P' in S. Then Lemma 5.15 and Lemma 5.8.1 on Page 75 imply that P is composed.

The final case is that Q is solely bound by sidepath-actions. Let P_2 be a path through C_2 in S with action o and Q_2 be the sidepath of P_2 through C with action o. If Q_2 is coreless then $pre(o) \cap C$ is a start and a stop, thus the lemma holds. Otherwise, sidepath Q_2 has a core and, according to Lemma 5.11.3 on Page 78, o lies during the core of Q_2. Let o_1 be the last action of the core of Q_2 before o and let o_2 be its first action afterwards. Let T' be the sequence of all path-actions of c-invariant C between o_1 and o_2 in S. Then $P' = (T', C)$ is a proper circular path in S, not necessarily regarding Π. According to Lemma 5.15 on the facing page, P' passes through a start and a stop, so we then know with Lemma 5.9 on Page 76 that $pre(o) \cap C$ is a start and a stop, too. (q. e. d.)

The preceding theorem is an important result that allows us to find path structures with solely relevant paths.

5.6.5 Type-3 Path Structures

We define type-3 path structures as a subset of type-2 path structures.

Definition 5.16 (Type-3 Path Structures)
A type-2 path structure Π is of type-3 *if all paths that can extend it (regarding type-2) are irrelevant. A path can* extend *a type-3 path structure Π if it can extend the type-2 path structure Π and is relevant. We call Π a* smallest *type-3 path structure if it comprises solely relevant paths.* ◇

We want to guarantee that there is a unique smallest type-3 path structure that covers a shortest solution. First, we show that the smallest type-3 path structure is unique.

Theorem 5.17 (Uniqueness of the Smallest Type-3 Path Structure)
The smallest type-3 path structure is unique for every planning problem. □

> **Proof:** Let Π and Π' be two smallest type-3 path structures of the same problem. Π is the result of extending the empty type-2 path structure by a list of paths and removing some of them later on. Let P_1 be the first extension that leads to Π. Therefore, P_1 is relevant and it connects a start of Π' and a stop of Π'. Then we know from Lemma 5.5 on Page 71 that the starts and stops resulting from P_1 are starts and stops of Π', whether or not P_1 is composed regarding Π'. We can use the same argument for all subsequent extending paths that lead to Π, such that all starts and stops of Π are also starts and stops of Π', respectively. We have chosen arbitrary path structures, so Π and Π' have the same starts and stops.
>
> As path relevancy is independent of a particular path structure and path minimality depends on the starts and stops only, a path is relevant and minimal regarding Π if and only if it is relevant and minimal regarding Π'. In other words, Π and Π' are the same. (q. e. d.)

We now show that the smallest type-3 path structure covers and is usable for a shortest solution to a planning problem. Although we know that minimal paths are free, the proof is not as easy as it seems: The problem is that replacing a path in a solution can introduce and eliminate starts and stops in that solution, thus the resulting sequence has other irrelevant paths and sidepaths than the original one.

We overcome this problem by splitting the replacements in two steps: First we eliminate disappearing sidepaths without replacing their paths, thus leaving other paths unchanged. This is done by introducing new actions that are not in the original problem. In the second step, we replace the new actions by those of the original problem. All these replacements yield plans whose execution in the initial state yields a state that entails the goal. As the final plan uses only actions of the problem, it is the anticipated solution.

Lemma 5.18 (Usability and Coverage of Type-3 Path Structures)
Let Ψ be a planning problem. Then there is a shortest solution S to Ψ such that the smallest complete type-3 path structure of Ψ is usable for S and covers S. □

> **Proof:** Let $\Psi = \big(\mathcal{O}, \mathcal{I}, \mathcal{G}\big)$. Let S be a shortest solution to Ψ. Let Π_3 be the smallest complete type-3 path structure of Ψ. We show the lemma in two steps: First we replace each minimal and irrelevant path P in S by a path P' that is the same as P but without its disappearing sidepaths. Then we

replace minimal paths in the resulting sequence by paths regarding Π_3, thus eliminating all new actions.

Note that these steps terminate: Each iteration of the first step strictly decreases the sum $\Sigma_i|\mathcal{C}_i|$, where o_i is the i-th action of S, $\mathcal{C}_i$ is a set of c-invariants, and a c-invariant C is in $\mathcal{C}_i$ if and only if o_i is a sidepath-action of C. Each iteration of the second step strictly decreases the number of actions that are not covered by Π_3. As both numbers are at least zero, the total number of iterations is finite. In detail, the steps are as follows.

Step 1: Let Π be the path structure with complete stop set that comprises all paths in Π_3 and all minimal paths regarding Π in S that are irrelevant. Therefore, Π is usable for S and Theorem 5.16 on Page 83 is applicable. In other words, minimal paths in S regarding Π are free.

Let $P = \big(\langle \ldots, o_i, \ldots \rangle, C\big)$ be a minimal and irrelevant path regarding Π in S. If there is no such path then go to Step 2.

Let P' be the path $\big(\langle \ldots, o_i', \ldots \rangle, C\big)$, where P_2 is a relevant path for P, $\mathcal{Q}$ is the set of sidepaths of P that have no corresponding sidepath in P_2, $\mathcal{F} = \bigcup_{(T,C)\in\mathcal{Q}} C$, o_i is the i-th action of P, and $o_i' = \big(pre(o_i)\backslash\mathcal{F}, add(o_i)\backslash\mathcal{F}, del(o_i)\backslash\mathcal{F}\big)$. In other words, P' is the path P without its disappearing sidepaths. Then S' is the sequence S where the i-th action of P is exchanged by the i-th action of P'. P is free in S, thus S' is a plan and the execution of S' in $\mathcal{I}$ yields a state that entails $\mathcal{G}$. We repeat Step 1 by taking S' as the new S.

Step 2: All minimal paths in S regarding Π are relevant, thus $\Pi = \Pi_3$ and Π_3 is usable for S. Let P be a minimal path in S that has an action o which is not covered by Π_3, i. e., that is not in $\mathcal{O}$ or that is not in a path in Π_3. If there is no such P then S is the final solution and the process stops.

Otherwise, let P' be a path in Π that can replace P and let S' be the sequence that is yielded by the replacement of P by P' in S. P is free in S, thus S' is a plan and the execution of S' in $\mathcal{I}$ yields a state that entails $\mathcal{G}$. We repeat Step 2 by taking S' as the new S. (q. e. d.)

With the last lemma, it is just a small step to show sufficiency.

Theorem 5.19 (Sufficiency of the Smallest Complete Type-3 Path Structure) *The smallest complete type-3 path structure suffices for every planning problem.* □

Proof: Follows directly from Theorem 5.2 on Page 66 and Lemma 5.18 on the preceding page. (q. e. d.)

We continue to show the same properties for path structures with arbitrary stop sets.

5.7 Arbitrary Stop Sets

In the previous sections, we introduce path structures with complete stop sets and the lemmas of uniqueness and sufficiency make use of this property. The drawback of complete

stop sets is that they can yield complete path structures with many unnecessary paths: Type-3 path structures guarantee a complete stop set by adding to the stop set all facts of c-invariants that are not mentioned by the goal. As the state after executing a solution comprises at most one fact of these c-invariants, all paths that end at the other facts are possibly unnecessary.

We cannot assume that the lemmas of the previous sections hold for path structures that do not have a complete stop because they use the fact that for every core of a sidepath there exists a path that has this core as subsequence. If, for example, the beginning of a sidepath in a shortest solution is the last stop of a c-invariant C, i. e., no fact of C is in the goal, then a complete path structure does not necessarily comprise a path through C that ends in the solution during or after the sidepath. In other words, we cannot apply the lemmas developed for type-3 path structures.

In the following, we will see that arbitrary stop sets do not pose a problem. In fact, the same technique that finds sufficient fixed points by using complete stop sets finds sufficient fixed points with arbitrary stop sets, too. In other words, there is no need to develop additional techniques. We show this nice result in three steps: Firstly, we define path structures with unrestricted stop sets – we call them path structures of *type-4* – and show some of their basic properties, e. g., the uniqueness of the complete type-4 path structure. Then we modify the lemmas from the previous sections for their use with arbitrary stop sets. Finally, we show the sufficiency of the complete type-4 path structure.

5.7.1 Type-4 Path Structures

A type-4 path structure is similar to a type-2 one, with the only difference that it does not use a complete stop set.

Definition 5.17 (Type-4 Path Structures)
A path structure $\Pi = (\mathcal{P}, \mathcal{B}, \mathcal{E})$ of a planning problem Ψ is of type-4 *if at least one of the following three cases holds:*

1. *Π is the empty type-0 path structure, i. e., the path structure $(\emptyset, \mathcal{I} \cap \mathcal{F}_\Psi^\mathcal{C}, \mathcal{G}')$, where $\mathcal{G}'$ is the union of $\mathcal{G} \cap \mathcal{F}_\Psi^\mathcal{C}$ and $\mathcal{F}_\Psi^{through}(\emptyset)$.*
2. *$\Pi' = (\mathcal{P} \setminus \{P\}, \mathcal{B}', \mathcal{E}')$ is a type-4 path structure, where P is a path that can extend Π'. Furthermore, $\mathcal{B}$ is the union of $\mathcal{B}'$, all regular endings of (proper) sidepaths of P, and $\mathcal{F}_\Psi^{through}(P)$. Likewise, $\mathcal{E}$ is the union of $\mathcal{E}'$, all beginnings of (proper) sidepaths of P, and $\mathcal{F}_\Psi^{through}(P)$.*
3. *$\Pi' = (\mathcal{P} \cup \{P\}, \mathcal{B}, \mathcal{E})$ is a type-4 path structure, where $P \notin \mathcal{P}$ and P is composed regarding Π'.*

Hereby, a path can extend a type-4 path structure Π if it can extend the type-0 path structure Π and it is relevant, minimal regarding Π, and of length $< 2^{|\mathcal{F}_\Psi|}$. ◇

Recall that the beginning of a sidepath is the fact that it begins with. Likewise, the ending of a sidepath is the fact with which it ends.

By comparison with type-2 path structures we see that there is a unique smallest type-4 path structure and that this path structure is usable for a shortest solution.

Theorem 5.20 (Properties of the Smallest Complete Type-4 Path Structure)
For every planning problem Ψ there is a unique complete type-4 path structure Π. Furthermore, Π is usable for every shortest solution to Ψ. □

Proof: We show the lemma by constructing a planning problem Ψ' such that the process of finding a type-4 path structure Π for Ψ corresponds one-to-one to the process of finding a type-2 path structure for Ψ'. Furthermore, all solutions to Ψ are plans of Ψ'. Likewise, there is no plan of Ψ' that is not a plan of Ψ. Then the lemma follows from the uniqueness and usability of the smallest type-2 path structure. Note that it is irrelevant for uniqueness and usability that the planning problem Ψ' is unsolvable.

We construct $\Psi' = \left(\mathcal{O}', \mathcal{I}', \mathcal{G}'\right)$ from $\Psi = \left(\mathcal{O}, \mathcal{I}, \mathcal{G}\right)$ as follows: Let $\mathcal{C}$ be the set of all c-invariants $C \in \mathcal{C}_\Psi$ with $C \cap \mathcal{G} = \emptyset$ and $\mathcal{C}^+ = \left\{C^+ \;\middle|\; C \in \mathcal{C}, C^+ = C \cup \{f_C\}, f_C \text{ new fact}\right\}$. Then $\mathcal{O}^+$ is a set of new actions o_C, one for each c-invariant $C \in \mathcal{C}$, such that o_C consumes the new fact f_C of C and adds the fact $\mathcal{I} \cap C$. We set $\mathcal{O}' = \mathcal{O} \cup \mathcal{O}^+$, $\mathcal{I}' = \mathcal{I}$, and $\mathcal{G}' = \mathcal{G} \cup \mathcal{F}^+$, where $\mathcal{F}^+$ is the set $\left\{f_C \;\middle|\; C \in \mathcal{C}\right\}$.

We now construct in parallel a type-4 path structure Π^4 for Ψ and the smallest type-2 path structure Π^2 for Ψ'. Let Π^4_i and Π^2_i be the respective path structures after the i-th simultaneous extension. Clearly, if a path P can extend Π^2_i then it can also extend Π^4_i. Furthermore, no fact in $\mathcal{F}^+$ is in $\mathcal{I}$ and none of the actions in $\mathcal{O}'$ adds a fact of $\mathcal{F}^+$, thus no path in Π^4 begins with a fact in $\mathcal{F}^+$. Consequently, if P can extend Π^4_i then it can also extend Π^2_i. Therefore Π^4_i is always the same as Π^2_i, minus the (unused) additional stops in $\mathcal{F}^+$.

As Π^2 is unique and usable for every shortest solution to Ψ, the same is true for Π^4. (q. e. d.)

Path structures can still comprise paths through a c-invariant C that is not mentioned by the goal. For example, if another path has a sidepath through C then the beginning of that sidepath is a stop of C. We will see that the last such stop is of importance for our examination of type-4 path structures, especially the last action in a solution that has a stop of C as precondition; we call it the *stop action* of C.

Definition 5.18 (Stop Actions)
Let $\Psi = \left(\mathcal{O}, \mathcal{I}, \mathcal{G}\right)$ be a planning problem, Π be a path structure of Ψ, S be a solution to Ψ, and C a c-invariant such that a sidepath-action of C is in S. If $C \cap \mathcal{G} = \emptyset$ then the stop action *of C in S is the last sidepath-action o of C in S that has a stop of C as precondition. If $C \cap \mathcal{G} \neq \emptyset$ then we say that the stop action of C is* after *S.* ◇

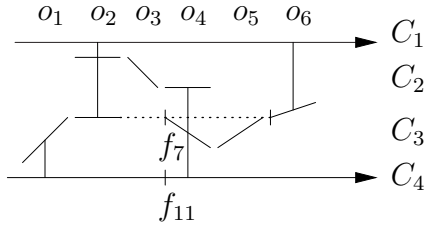
If there is no stop action of a c-invariant in or after a solution then there is no sidepath through that c-invariant either. Note that we motivation the convention of saying "the stop action of C is after S" by the view of the goal as an action that has the precondition $\mathcal{G}$.

Similar to stop actions, we are interested in the fact of a c-invariant C that is the last stop of C in a solution.

Example 5.6 (Construction of a Type-4 Path Structure)
Let us construct the smallest complete type-4 path structure for the planning problem $\Psi = \big(\{o_1, \ldots, o_6\}, \{f_1, f_4, f_6, f_{10}\}, \{f_3, f_5, f_{12}\}\big)$. *We have already seen this problem in Example 5.4 on Page 73, where we found its smallest complete type-2 path structure. The actions and c-invariants of* Ψ *are as follows.*

c-invariants	o_1	o_2	o_3	o_4	o_5	o_6
$C_1 = \{f_1, f_2, f_3\}$		$f_1 \rightarrow f_2$				$f_2 \rightarrow f_3$
$C_2 = \{f_4, f_5\}$		f_4	$f_4 \rightarrow f_5$	f_5		
$C_3 = \{f_6, f_7, f_8, f_9\}$	$f_6 \rightarrow f_7$	f_7		$f_7 \rightarrow f_8$	$f_8 \rightarrow f_7$	$f_7 \rightarrow f_9$
$C_4 = \{f_{10}, f_{11}, f_{12}\}$	$f_{10} \rightarrow f_{11}$			$f_{11} \rightarrow f_{12}$		

The following graph depicts the only solution to the problem.



In the following, we construct the complete type-4 path structure for this planning problem. We show the path structures $\Pi_i = \big(\mathcal{P}_i, \mathcal{B}_i, \mathcal{E}_i\big)$, *where* Π_0 *is the empty path structure and* Π_{i+1} *is the path structure after connecting all starts and stops present in path structure* Π_i. *We denote paths with* $P^i_{p,q} = \big(T^i_{p,q}, C_p\big)$, *where* i *refers to the path structure,* p *is the index of the c-invariant and* q *is an index of the path.*

1. *We start with the path structure* Π_0, *where* $\mathcal{P}_0 = \emptyset$, $\mathcal{B}_0 = \mathcal{I}$ *and* $\mathcal{E}_0 = \mathcal{G}$.

2. *In our first iteration, we add all paths that go from the initial state to the goal. This yields the path structure* Π_1 *with* $\mathcal{P}_1 = \{P^1_{1,1}, P^1_{2,1}, P^1_{4,1}\}$, *where* $T^1_{1,1} = \langle o_2, o_6\rangle$, $T^1_{2,1} = \langle o_3\rangle$, *and* $T^1_{4,1} = \langle o_1, o_4\rangle$. *Consequently,* $\mathcal{B}_1 = \mathcal{B}_0 \cup \{f_5, f_8, f_9\}$ *and* $\mathcal{E}_1 = \mathcal{E}_0 \cup \{f_4, f_5, f_6, f_7\}$.

3. *Then we add the paths with sequences* $T^2_{3,1} = \langle o_1\rangle$, $T^2_{3,2} = \langle o_1, o_4, o_5\rangle$, $T^2_{3,3} = \langle o_5\rangle$, *and* $T^2_{3,4} = \langle o_5, o_4, o_5\rangle$. *Therefore,* $\mathcal{B}_2 = \mathcal{B}_1 \cup \{f_{11}, f_{12}\}$ *and* $\mathcal{E}_2 = \mathcal{E}_1 \cup \{f_{10}, f_{11}\}$.

4. *In the third iteration, we add the path* $P^3_{4,1} = \big(\langle o_1\rangle, C_4\big)$, $P^3_{4,2} = \big(\langle o_4\rangle, C_4\big)$, *which yields* $\mathcal{B}_3 = \mathcal{B}_2 \cup \{f_7\}$ *and* $\mathcal{E}_3 = \mathcal{E}_2$. *We disregard the composed path* $P^1_{4,1}$, $P^2_{3,2}$, *and* $P^2_{3,4}$.

5. *Finally, we add the path* $P^4_{3,1} = \big(\langle o_4, o_5\rangle, C_4\big)$, *which yields* $\mathcal{B}_4 = \mathcal{B}_3$ *and* $\mathcal{E}_4 = \mathcal{E}_3$, *so that the process reaches a fixed point.*

The solution S *is covered, e. g., by the set* $\big\{\big(\langle o_2, o_6\rangle, C_1\big), \big(\langle o_3\rangle, C_2\big), \big(\langle o_1\rangle, C_3\big), \big(\langle o_5\rangle, C_3\big), \big(\langle o_4\rangle, C_4\big)\big\}$ *of minimal paths. By comparison with Example 5.4 on Page 73 we see that for this problem the complete path structure of type-4 is smaller than the smallest complete one of type-2.* ■

Definition 5.19 (The Last Stop of a c-Invariant)
Let $(\mathcal{O}, \mathcal{I}, \mathcal{G})$ be a planning problem, Π be a path structure, and S be a solution to Ψ. Then the last stop *f of a c-invariant C in S is defined as follows: If the stop action o of C is in S then $f \in pre(o) \cap C$. If the stop action of C is after S then f is the fact of C in the goal, i. e., $f \in \mathcal{G} \cap C$. Otherwise, f is the fact of C in the initial state, i. e., $f \in \mathcal{I} \cap C$.* $\diamond$

We can classify sidepaths according to their location in respect to their last stop: Either a sidepath Q ends before its last stop, it begins afterwards, or it is neither before nor after its last stop. We will later see that for sidepaths of minimal paths always one of the first two cases holds.

Definition 5.20 (The Location of Sidepaths in Relation to Their Last Stop)
Let $\Psi = (\mathcal{O}, \mathcal{I}, \mathcal{G})$ be a planning problem, S a solution to Ψ, and Π a path structure of Ψ. Let C be a c-invariant, o the stop action of C in S, and Q a sidepath through C of a path in S.

Sidepath Q is before its last stop *if either one of the following three cases holds: (1) Q ends before o in S. (2) Action o is during Q but not in Q. Furthermore, if Q has a core then o is after the core of Q. (3) Action o is the last action of Q and a proper sidepath-action of C.*

Sidepath Q is after its last stop *if at least one of the following two cases holds: (1) Q begins after o. (2) o is in Q and is a path-action of C.* $\diamond$

The following section shows that sidepaths of the first class can be handled as if they result from a path structure with a complete stop set. For example, if all sidepaths of a minimal path are before their stops then the path is free.

5.7.2 The Lemmas for Path Structures with Arbitrary Stop Sets

In the previous sections, we have formulated lemmas on path structures with a complete stop sets. In particular, Lemmas 5.8 to 5.14 on Pages 75—81 allowed us to show the sufficiency of the smallest complete type-3 path structure. We now see that we can formulate similar lemmas for type-4 path structures if we prohibit their application to sidepaths after their last stops. In order to simplify the presentation and to avoid an unnecessary duplication, we only state the new lemmas together with their additional prerequisites. Their proofs are similar to the proofs of the original lemmas, thus we explain the necessary modifications to get the new proofs.

Lemma 5.21 (Lemmas 5.8 to 5.14 for Arbitrary Stop Sets)
Let Ψ be a planning problem, S a shortest solution to Ψ, and Π a path structure of Ψ with arbitrary stop set that is usable for S. Then the following lemmas (originally formulated for path structures with complete stop sets) hold:

- ***Lemma 5.8:*** *A path P_1 in S regarding Π is composed if at least one of the following is true:*

  1. *P_1 has a sidepath through C_2 and there is a start and a stop of C_2 in S between two adjacent path-actions o_1 and o_2 of C_2 in P_1.* ***Additional prerequisite:*** *The stop action of C_2 in S is after o_2.*

2. *P_1 has two or more sidepaths with core through the same c-invariant C_2.* ***Additional prerequisite:*** *Let Q_2 be the second such sidepath and o the first action of the core of Q_2. Then the stop action of C_2 is after o in S.*

- ***Lemma 5.9:*** *Let $P_1 = \big(\langle \ldots, o_1, \ldots, o, \ldots, o_2, \ldots \rangle, C_1\big)$ be a path in S regarding Π, where o is a proper sidepath-action of C_2, o_1 the last path-action of C_2 before o in P_1, and o_2 the first path-action of C_2 after o in P_1. If there is a start and a stop of C_2 in S between actions o_1 and o_2 then $pre(o) \cap C_2$ is a start and a stop.* ***Additional prerequisite:*** *The stop action of C_2 in S is after action o_2.*

- ***Lemma 5.10*** *holds for path structures in general.*

- ***Lemma 5.11:*** *Let o be an action in plan S and P_1 a shortest path in S regarding Π with action o. If P_1 has a sidepath Q_1 with core through C_2 then the following is true:*

  1. *If the core of Q_1 begins before o then there is no stop of C_2 in S between the first action of the core of Q_1 and action o.* ***No additional prerequisite necessary.***
  2. *If the core of Q_1 ends after o then there is no start of C_2 in S between o and the last action of the core of Q_1.* ***Additional prerequisite:*** *The start of C_2 is before the last stop of C_2 in S. In other words, Q_1 is before its last stop.*
  3. *Action o is during the core of Q_1.* ***Additional prerequisite:*** *The stop action of C in S is after the beginning of the core of Q_1. In other words, Q_1 is before the last stop of C_2 .*

- ***Lemma 5.12*** *holds for circular paths in general.*

- ***Lemma 5.13:*** *Let P be a circular path in solution S that is not necessarily a path regarding path structure Π. If path P has a non-circular sidepath Q_2 then P passes through a start and a stop.* ***Additional prerequisite:*** *P ends before its stop action in S and Q_2 is before its last stop.*

- ***Lemma 5.14:*** *Let P be a circular path in S, not necessarily regarding Π. If the non-proper sidepath of P is bound then P is composed.* ***Additional prerequisite:*** *P ends before its stop action in S. Furthermore, if a sidepath of a path in S regarding Π is bound by an action o that is in $\mathcal{O}^{\mathcal{C}}$ then there is a c-invariant C_2 so that o is a path-action of C_2 and o is before the last stop of C_2.*

  Proof: The proofs follow the respective proofs of Lemmas 5.8 to 5.14 on Pages 75—81. The difficulty in adopting these proofs lies in sidepaths through c-invariants C with $C \cap \mathcal{G} = \emptyset$: A usable path structure Π with complete stop set guarantees that whenever a path in S regarding Π has a sidepath Q through C with core then Q lies during a path through C in S regarding Π. This is not the case for usable type-4 path structures.

  The additional prerequisites guarantee the existence of all those paths that are necessary for their respective proofs. If a sidepath is not mentioned in the additional prerequisites then its existence is guaranteed otherwise, e. g., because

it is during a path regarding Π through the same c-invariant or because there is a later sidepath through the same c-invariant.

The second additional prerequisite of Lemma 5.14 on Page 81 guarantees that all path-actions which bind sidepaths of path P are in a path in S that is a path regarding Π. The proof to Lemma 5.24 on Page 93 will also show that this additional prerequisite is always satisfied. We will use the generalized version of Lemma 5.14 only after the proof of Lemma 5.24. (q. e. d.)

We will use such a new lemma only if it is guaranteed that its additional requirement is fulfilled.

The following two lemmas formulate important observations about sidepaths and their last stop in a solution. The first one states that the stop action of C is in all sidepaths through C that are after their last stop. In other words, all sidepaths through a given c-invariant after their last stop have common actions.

Lemma 5.22 (Stop Action of C is in Sidepath through C after Its Last Stop) *Let $\Psi = (\mathcal{O}, \mathcal{I}, \mathcal{G})$ be a planning problem, S a solution to Ψ, and Π path structure of Ψ that is usable for S. Let C be a c-invariant with $C \cap \mathcal{G} = \emptyset$, o the stop action of C in S, and o' a sidepath-action of C in S after o. If P is a shortest path in S regarding Π with action o' then (1) o is an action in P and (2) both actions are in the same sidepath of P through C.* □

Proof: Let o' be in sidepath Q through C of P, thus Q ends after o in S. (1) If o is not in P and the core of Q begins before o in S then Lemma 5.11.1 on Page 78 states that P is not a shortest path with action o'. If o is not in P and either Q is coreless or its core begins after o then o is not the stop action of C in S. The only case left is that o is in P. (2) If Q begins after o then o is not the stop action of C in S. Therefore, Q begins before o or o is the first action of Q. With (1) follows that o is in Q. (q. e. d.)

Furthermore, all sidepaths of a minimal path are either before or after their respective last stops.

Lemma 5.23 (Sidepath of Minimal Path either before or after Its Last Stop) *Let $\Psi = (\mathcal{O}, \mathcal{I}, \mathcal{G})$ be a planning problem, S a solution to Ψ, and Π a path structure of Ψ that is usable for S. If P is a minimal path in S regarding Π then every sidepath of P is either before or after its last stop.* □

Proof: Let Q be a sidepath through C of P and let o be the stop action of C in S. If Q ends before o or begins after this action then the lemma holds. Otherwise, o is during Q.

If o is not in Q then either Q is coreless or o is during the right fringe of Q, thus Q is before its last stop: As the core of Q begins with a stop, o cannot be before the core. If o is during the core then a postfix P' of P is a path regarding Π, too. P' has a sidepath through C with core that begins later than o. In both cases, o is not the stop action of C in S.

If o is in Q and Q is coreless then o is not in a proper prefix of Q. In other words, o is the last action of Q and Q is before its last stop.

If o is in Q and Q has a core then o is not in the left fringe of Q. If it is in the core then it must be its first action, thus Q is after its last stop. If it is in the right fringe of Q then it must be its last action. In this case, Q is before its last stop.

These are all cases, so the lemma holds. (q. e. d.)

We now show that sidepaths of minimal paths are free if they are after their stop action. This observation leads directly to the sufficiency of type-4 path structures.

5.7.3 Dangling Actions and Dangle Situations

We now show that sidepaths after their last stops are free. Assume they were not. Then they are bound by actions after their last stop, called *dangling actions*.

Definition 5.21 (Dangling Actions)
Let $\Psi = (\mathcal{O}, \mathcal{I}, \mathcal{G})$ be a planning problem, S be a solution to Ψ, and Π be a path structure of Ψ. Then we say that an action o in S dangles *in C if o is after the stop action of C in S and binds a sidepath through C.* □

We now show that there are no dangling actions in shortest solutions. In other words, all sidepaths of minimal paths that are after their last stop are free. Firstly, we will see that we can characterize a certain configuration of actions and paths in that solution, connected with a dangle action; we call it the *dangle situation* of o. We then show that we can remove all dangling actions from the solution together with their dangle paths such that the resulting sequence is another solution. Obviously, this means that a shortest solution does not have any dangling actions. We begin by defining dangling situations.

Definition 5.22 (Dangle Situations)
Let Ψ be a planning problem, S a solution to Ψ, and Π a path structure. Let o be a dangling action in S such that o is in a path in S regarding Π. Then a dangle situation *of o in S regarding Π is a seven-tuple $(C, P_1, P_2, o_1, o_2, o_3, P)$, where o dangles in c-invariant C, the paths P_1, P_2, P are in S, the actions o, o_1, o_2, o_3 are in S, and the following holds:*

1. *o is a path-action of c-invariant C_2. P_2 is a shortest path through C_2 with action o.*
2. *o_1 is the stop action of C. P_1 is a shortest path through C_1 with action o_1.*
3. *o_2 is the last common action of P_1 and P_2 before o that is a path-action of C.*
4. *o_3' is the first common action of P_1 and P_2 after the last sidepath-action of C in P_2. o_3 is the last common action of P_1 and P_2 with $pre(o_3) \cap C_2 = pre(o_3') \cap C_2$.*
5. *P is the longest subsequence of P_2 between o_2 and o_3. Furthermore, P is circular.*
6. *Paths P_1, P_2, and P are distinct. Likewise, actions o, o_1, and o_3 are distinct, as well as actions o, o_2, and o_3.*

We call path P the dangle path *of o.* ◇

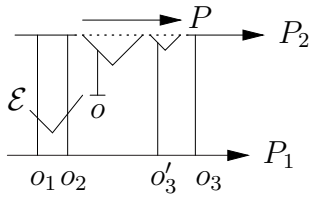


Figure 5.5: A dangle situation $\big(C, P_1, P_2, o_1, o_2, o_3, P\big)$ of action o. Hereby, o dangles in C, o_1 is the stop action of C, and P is the dangle path of o.

A dangle situation is depicted in Figure 5.5. We now show that if path structure Π is usable for S then a dangle situation regarding Π exists for every dangle action in S.

Lemma 5.24 (For every Dangle Action there is a Dangle Situation) *Let $\Psi = \big(\mathcal{O}, \mathcal{I}, \mathcal{G}\big)$ be a planning problem, S a shortest solution to Ψ, and Π a path structure of Ψ that is usable for S. Then there is a dangling situation $(C, P_1, P_2, o_1, o_2, o_3, P)$ for every dangling action in S regarding Π.* □

Proof: We first show that every dangling action o in S is in a path in S regarding Π, i. e., that o is a path action of a c-invariant C_2 and the stop action of C_2 in S is after o. We then show that there is a dangling situation for o in S as asserted.

Let us assume that there is a dangling action in S that is not in any path in P regarding Π. Then o does not add any fact in $\mathcal{F}^{\overline{\mathcal{C}}}_{\Psi}$ that is needed by a later action of the goal. Therefore, o is a path-action and it adds a necessary fact in $\mathcal{F}^{\mathcal{C}}_{\Psi}$. Let S' be the sequence S without all such actions o. As S' is not a solution to Ψ, there is a path P^* in S with a sidepath Q^* and the beginning of Q^* is false in S'. Furthermore, Q^* begins after its stop action. As P^* is a path in Π, the beginning of Q^* is a stop and o is not after its stop action. This is a contradiction, so that no such dangling action exists.

Now we show that there is a dangling situation for every dangling action.

1. We just have seen that all dangling actions are path-actions and they are in a path in S regarding Π. P_2 is one of these paths.
2. The existence of o_1 and P_1 follows from the definition of dangling actions.
3. From Lemma 5.22 on Page 91 we know that o_1 is a common action of P_1 and P_2 before o. Furthermore, o_1 is a path-action of C because there are sidepath-actions of C after o_1 in S. Then o_2 exists as asserted.
4. P_1 has a sidepath through C_2 with a core that begins before o and we know from Lemma 5.11.3 on Page 78 that the core ends after o. Then actions o_3 and o'_3 exist as asserted.
5. Action o is in P, so P is proper. As actions o_2 and o'_3 are adjacent actions in the induced path of Q_1, $pre(o_2) \cap C_2 = pre(o'_3) \cap C_2$ and P is circular.
6. o binds P_1, thus P_1 and P_2 as well as P_1 and P are distinct. The same holds for P and P_2, as the former is shorter than the latter. Actions o_1, o, and o_3 are distinct because they are subsequent actions in S. For the same reason o_2, o, and o_3 are distinct. (q. e. d.)

Note that o_1 and o_2 as well as o_3' and o_3 may or may not be the same action. Also note that the first part of the preceding lemma implies that the second additional prerequisite of the generalized version of Lemma 5.14 on Page 81 is always satisfied (cf. Page 90).

The following two lemmas prepare the removal of dangle paths. Each of them allows to eliminate some reasons why the removal could yield a conflicting sequence. For example, if the paths in a dangling situation would be bound then they would pass through starts.

Lemma 5.25 (No Starts during Dangling Situations)
Let Ψ be a planning problem, S a solution to Ψ, and Π the a path structure of Ψ that is usable for S. Let $\big(C, P_1, P_2, o_1, o_2, o_3, P\big)$ be a dangle situation of action o in S. Then neither of P_1 and P_2 passes through a start of its respective c-invariant between o_2 and o_3 in S. □

> **Proof:** If P_1 passes through a start of C_1 between o_2 and o_3 in S then a proper postfix P_1' of P_1 is a path regarding Π and P_1' has a sidepath through C_2 that begins with $pre(o_3) \cap C_2$. Therefore, P_2 is not the shortest path through C_2 with action o. A similar argument contradicts the assumption that P_2 passes through a start between o_2 and o_3 in S. (q. e. d.)

We will remove dangling actions together with their dangling path. These paths are circular, so after the removal of a single dangle path through c-invariant C, the resulting sequence of path-actions of C is another path through C. It is not trivial to see the same for the combined removal of two or more dangling paths: If dangle paths overlap then their combined sequence can be non-circular, so that their removal yields a conflict. The following lemma assures us that this case does not occur.

Lemma 5.26 (Dangle Paths do not Partially Overlap)
Let Ψ be a planning problem, S a solution to Ψ, and Π a path structure of Ψ that is usable for S. Let $\big(C, P_1, P_2, o_1, o_2, o_3, P\big)$ and $\big(C, P_1', P_2', o_1, o_2', o_3', P'\big)$ be two dangle situations of actions o and o' in S, respectively. If the dangle paths P and P' have a common action then one of them is a subsequence of the other. □

> **Proof:** We proof the lemma by contradiction. Let us assume that P begins before the beginning and ends before the ending of P'. Furthermore, both paths have a common action. Consequently, o_3 is an action in P'. Let P_2^* be the path P_2 without the actions of path P'. Then o_2 is the last path-action of a sidepath of P_2^* through C. Π is usable for S, so it covers P_2^* and $post(o_2) \cap C_1$ is a start, where P_1 goes through C_1. With Lemma 5.25 this contradicts our assumption. (q. e. d.)

These preparations enable us to show that a shortest solution does not have dangling actions. This fact follows from the argument that for every solution with dangle path there is a shorter solution without that path. The problem with removing dangle paths is that they can have bound sidepaths. We will see that if we remove all dangle paths at once then these bindings do not matter.

Lemma 5.27 (Shortest Solutions do not have Dangling Actions)
Let $\Psi = \big(\mathcal{O}, \mathcal{I}, \mathcal{G}\big)$ be a planning problem, S a shortest solution to Ψ, and Π a path structure of Ψ that is usable for S. Then there are no dangling actions in S. □

Proof: We prove the lemma by contradiction. We assume that there are dangling actions in S and show that this implies a solution to Ψ that is shorter than S. Let T be the partial sequence of S that comprises all dangle paths in S and only those paths. Let S' be the sequence S without the actions in T. According to our assumption, S' is not a solution to Ψ. Note that we use all lemmas in their version for arbitrary stop sets, as given on Page 89.

What are possible reasons why S' is not a solution to Ψ? Lemma 5.12 on Page 79 gives four reasons why a circular path P is in a shortest solution: (1) The non-proper sidepath of path P is bound. (2) Path P has a proper sidepath Q that is non-circular. (3) Path P has a proper sidepath Q that is bound. (4) An action in path P adds a fact in $\mathcal{F}_\Psi^{\overline{C}}$. If any of these cases applies to a dangle path then S' might not be a solution to Ψ. In our case, there can be a further reason: (5) Each single dangle path can be removed on its own but the combination of all dangle paths cannot. We now show that each of the five pints yields a contradiction.

1. P ends before its last stop, so the version of Lemma 5.14 for arbitrary stop sets is applicable and the non-proper sidepath of P is free (cf. Page 81).
2. Here, we note that, because there is no dangling action in S', if Q is a sidepath after its last stop then Q can be safely removed. Otherwise, Q is before its last stop. As dangle paths do not pass through a start, Q is circular because the version of Lemma 5.13 for arbitrary stop sets is applicable (cf. Page 80).
3. If Q is before its last stop then the version of Lemma 5.14 for arbitrary stop sets is applicable and Q is free; if it is after its last stop then it is free because there are no dangling actions in S' (cf. Page 81).
4. This case does not occur because P would pass through a start.
5. Let us first consider the case of the combined removal of two dangle paths P and P' through the same c-invariant C in S. Let P_1 be the longest path through C in S. From Lemma 5.26 on the preceding page we know that either P and P' have no action in common or one of them is a subsequence of the other. Therefore, their combined removal from P_1 yields another path, and this path begins and ends with the same fact as P_1, respectively. Then the same is true for the removal of any combination of dangle paths in S from P_1.

We have eliminated all possible reasons why S' might not be a solution to Ψ. Thus, S' is a shorter solution to Ψ than S, which contradicts our initial assumption and the lemma holds. (q. e. d.)

With the absence of dangling actions, it is easy to show that all minimal paths in a shortest solution are free.

Theorem 5.28 (Minimal Paths are Free)
Let Ψ be a planning problem, S a shortest solution to Ψ, and Π a path structure of Ψ that is usable for S. Then all minimal paths in S regarding Π are free. □

> **Proof:** We now know that a sidepath is either before or after its last stop and in the second case it is free. This allows us to formulate a version of Lemma 5.15 on Page 82 for path-structures with arbitrary stop sets which has the additional prerequisite that the circular path ends before its last stop. We then can formulate a corresponding version of Lemma 5.16 on Page 83. (q. e. d.)

5.7.4 Type-5 Path Structures

What is left is the definition of a type-5 structure similar to the one of type-3 and to show that the smallest complete type-5 path structure suffices.

Definition 5.23 (Type-5 Path Structures)
A type-4 path structure Π is of type-5 *if all paths that can extend it (regarding type-4) are irrelevant. A path can* extend *a type-5 path structure Π if it can extend the type-4 path structure Π and is relevant. We call Π a* smallest *type-5 path structure if it comprises solely relevant paths.* ◇

The necessary lemmas and proofs are similar to their type-3 variants. In order to avoid an unnecessary duplication, we do not repeat these arguments here.

Theorem 5.29 (Sufficiency of the Complete Type-5 Path Structure)
The smallest complete type-5 path structure suffices for every planning problem. □

> **Proof:** We show that the complete type-5 path structure covers a shortest solution to the problem in the same way than we showed this for the smallest complete type-3 path structure in Section 5.6.5 on Page 83. (q. e. d.)

The previous theorem concludes our development of path reduction. The following chapter elaborates on implementation details and demonstrates the results of applying path reduction to planning problems.

Chapter 6

Implementation and Experiments

This chapter describes our implementation of path reduction. First, we present the algorithm of path reduction in ground representation before we describe how to lift it to the general level. Then we explain the design of our implementation before we give experimental results.

6.1 The Algorithm of Path Reduction

The previous chapter shows that the smallest complete type-5 structure suffices to solve planning problems. In other words, if the problem is solvable then it has a solution that is solely composed by actions of the paths in the smallest complete type-5 path structure, together with their necessary actions of the unstructured part of the problem. We now present the algorithm of path reduction, i. e., an algorithm to find this sufficient action set for a given planning problem.

6.1.1 Type-6 Path Structures

The algorithm of path reduction operates on path structures of *type-6*, which differ from the types presented in Chapter 5 in two respects. The first difference is that a type-6 path structure includes paths that do not connect one of its starts with one of its stops. In contrast, all paths in a type-5 path structure Π are paths regarding Π.

Definition 6.1 (Type-6 Path Structures)
A type-6 *path structure* $\Pi = \big(\mathcal{P}, \mathcal{B}, \mathcal{E}\big)$ *of a planning problem* $\Psi = \big(\mathcal{O}, \mathcal{I}, \mathcal{G}\big)$ *has each of the following properties:*

1. $\mathcal{P}$ *is a set of relevant paths of* Ψ.
2. $\mathcal{B}$ *is the union of* $\mathcal{I}$, $\big\{f \;\big|\; P \in \mathcal{P}', f$ *is a regular ending of a proper sidepath of* $P\big\}$, *and* $\mathcal{F}_{\Psi}^{through}(\mathcal{P}')$.
3. $\mathcal{E}$ *is the union of* $\mathcal{G}$, $\big\{f \;\big|\; P \in \mathcal{P}', f$ *is a beginning of a proper sidepath of* $P\big\}$, *and* $\mathcal{F}_{\Psi}^{through}(\mathcal{P}')$.

Hereby, $\mathcal{P}'$ *is a subset of* $\mathcal{P}$ *such that each path* $P \in \mathcal{P}'$ *is a path regarding* Π. ◇

The second difference is the notion of completeness of a type-6 path structure, which depends on its *limit*, i. e., the length of its longest paths.

Definition 6.2 (Completeness of a Type-6 Path Structures)
Let $n \geq 0$ be an integer. A path structure Π has the limit *n if all paths in Π are of length n or shorter.*

A type-6 path structure Π with limit i is complete *if each of the following properties holds:*

1. *Every path P that can extend Π is either irrelevant, composed regarding Π, or of length $i+1$ or longer.*
2. *If a path P in Π is a path regarding Π then $\mathcal{B}$ comprises the union of all regular endings of proper sidepaths of P and $\mathcal{F}_{\Psi}^{through}(P)$.*
3. *If a path P in Π is a path regarding Π then $\mathcal{E}$ comprises the union of all beginnings of proper sidepaths of P and $\mathcal{F}_{\Psi}^{through}(P)$.*

Hereby, a path can extend *a type-6 path structure Π if it is not in Π.* ◇

In other words, a complete type-6 path structure with limit i for a planning problem Ψ comprises all minimal and relevant paths of Ψ that are of length i or shorter. In addition, if $i = 2^{|\mathcal{F}_\Psi|} - 1$ then it has the same sets of starts and the same sets of stops as the complete type-5 path structure for the same problem (cf. Section 6.1.7 on Page 103). Note that the limit of a complete type-6 path structure can be smaller than the longest path in the smallest complete type-5 path structure. For reasons of brevity, we refer to the smallest complete type-5 path structure simply by the complete type-5 path structure from now on.

6.1.2 The Function *path_reduction*

The top-level function of path reduction, which is given in Figure 6.1 on the facing page, takes as input a planning problem and a set of c-invariants for that problem; the output is a reduced planning problem. In short, *path_reduction* computes a sequence of complete type-6 path structures with increasing limit, until one of them meets a termination criterion; it then uses the final path structure to build the reduced planning problem.

Function *path_reduction* calls four subfunctions, which we explain in the following sections:

- *get_path_structure_with_limit_1* initializes *path_reduction* by constructing the complete type-6 path structure with limit 1.
- *get_paths_of_length* computes all paths with length i that are in a complete type-6 path structure with limit i.
- *find_completion* finds all new starts and stops that result from these additional paths.
- *get_be_from_paths* computes all starts and stops that result from a set of paths in combination with a given set of starts and a set of stops.

Input: Ψ – a planning problem
$\mathcal{C}$ – a set of c-invariants
Output: Ψ_{res} – a reduced planning problem

funct *path_reduction*$(\Psi, \mathcal{C})$
$(\mathcal{P}, \mathcal{B}, \mathcal{E}) \coloneqq$ *get_path_structure_with_limit_1*$(\Psi, \mathcal{C})$
$n \coloneqq 2 \cdot max\{s \mid C \in \mathcal{C},\ s \text{ size of } C\}$

for $i \coloneqq 1$ **to** n **do**
$\mathcal{P}_{new} \coloneqq$ *get_paths_of_length*$(i{+}1,\ (\mathcal{P}, \mathcal{B}, \mathcal{E}))$
if $(\mathcal{P}_{new} = \emptyset)$
then exit for
else $(\mathcal{B}_{new}, \mathcal{E}_{new}) \coloneqq$ *get_be_from_paths*$(\mathcal{P}_{new},\ \mathcal{B},\ \mathcal{E})$
$\mathcal{B}_{ext} \coloneqq \mathcal{B}_{new} \setminus \mathcal{B}$
$\mathcal{E}_{ext} \coloneqq \mathcal{E}_{new} \setminus \mathcal{E}$
$\mathcal{P}_{next} \coloneqq \mathcal{P} \cup \mathcal{P}_{new}$
$(\mathcal{B}_{next}, \mathcal{E}_{next}) \coloneqq$ *find_completion*$((\mathcal{P}_{next}, \mathcal{B}, \mathcal{E}),\ \mathcal{B}_{ext},\ \mathcal{E}_{ext})$
$(\mathcal{P}, \mathcal{B}, \mathcal{E}) \coloneqq (\mathcal{P}_{next}, \mathcal{B}_{next}, \mathcal{E}_{next})$
fi
od

$\mathcal{O} \coloneqq \{o \mid P \in \mathcal{P},\ f_b \in \mathcal{B},\ f_e \in \mathcal{E},\ P \text{ connects } f_b \text{ to } f_e,\ o \text{ action in } P\}$
$\mathcal{O}_{res} \coloneqq \mathcal{O} \cup \mathcal{O}_{\Psi}^{nec}(\mathcal{O})$
$\Psi_{res} \coloneqq (\mathcal{O}_{res}, \mathcal{I}, \mathcal{G})$
return Ψ_{res}
.

Figure 6.1: The main function *path_reduction*.

The main part of *path_reduction* is a loop that incrementally computes a sequence of complete type-6 path structures with increasing limit: the input to the i-th iteration of the loop is the one with limit i and result is the one with limit $i + 1$. First, the loop computes the set $\mathcal{P}_{new}$ of all minimal and relevant paths of length $i + 1$. Then, it finds all additional starts and stops that result from these paths. The loop terminates in iteration l if either it cannot find any new path of length $l + 1$ or if l equals twice the size of the largest c-invariant plus one. We justify these criteria in Section 6.1.8 on Page 104.

The remaining steps of *path_reduction* compute the reduced planning problem: The result of the final iteration has the same starts and stops as the complete type-5 path structure for the same problem. The algorithm computes all actions from paths that connect such starts and stops. It then adds all necessary actions from the unstructured part and returns the reduced planning problem.

6.1.3 The Complete Type-6 Path Structure with Limit One

The function *get_path_structure_with_limit_1* computes the complete type-6 path structure with limit 1. First, it constructs two sets of paths: the set of all empty paths and the set

Input: Ψ – a planning problem
$\mathcal{C}$ – a set of c-invariants
Output: $(\mathcal{P}, \mathcal{B}, \mathcal{E})$ – the type-6 path structure with limit 1 for Ψ

funct *get_path_structure_with_limit_1*$(\Psi, \mathcal{C})$
$(\mathcal{O}, \mathcal{I}, \mathcal{G}) \coloneqq \Psi$
$\mathcal{P}_0 \coloneqq \{(\langle\,\rangle, C) \mid C \in \mathcal{C}\}$
$\mathcal{P}_1 \coloneqq \{(\langle o \rangle, C) \mid o \in \mathcal{O},\ C \in \mathcal{C},\ o \text{ is path-action of } C\} \cup \mathcal{P}_0$
$\mathcal{P} \coloneqq$ *get_paths_of_length*$(1,\ (\mathcal{P}_1, \emptyset, \emptyset)) \cup \mathcal{P}_0$
$\mathcal{B}_0 \coloneqq \mathcal{I} \cap \mathcal{F}^{\mathcal{C}} \cup \mathcal{F}_{\Psi}^{through}(\emptyset)$
$\mathcal{E}_0 \coloneqq \mathcal{G} \cap \mathcal{F}^{\mathcal{C}} \cup \mathcal{F}_{\Psi}^{through}(\emptyset)$
$(\mathcal{B}, \mathcal{E}) \coloneqq$ *find_completion*$((\mathcal{P}, \emptyset, \emptyset),\ \mathcal{B}_0,\ \mathcal{E}_0)$
return $(\mathcal{P}, \mathcal{B}, \mathcal{E})$
.

Figure 6.2: The function *get_path_structure_with_limit_1* computes the complete type-6 path structure with limit 1.

of all paths of length one. The call of *get_paths_of_length* results in those paths of length one that cannot be replaced. Together with the set of empty paths, these are the paths of the anticipated path structure.

The remaining commands find the necessary starts and stops. $\mathcal{B}_0$ and $\mathcal{E}_0$ are the sets of starts and stops that result directly from the initial state and the goal of the problem, respectively. Hereby, the function $\mathcal{F}_{\Psi}^{through}$, if called with an empty set, returns a set of facts that are relevant for the goal of Ψ (cf. Definition 3.12 on Page 33). The remaining starts and stops for the complete type-6 path structure with limit 1 are computed by the function *find_completion*. Note that the function $\mathcal{O}_{\Psi}^{nec}$ computes all actions in $\mathcal{O}_{\Psi}^{\overline{\mathcal{C}}}$ that are necessary for a set of path-actions (cf. Definition 3.11 on Page 33).

6.1.4 Extending the Limit: Additional Paths

Each iteration of the main loop of *path_reduction* incrementally computes a complete type-6 path structure with a limit that is increased by one. The additional paths are constructed by the function *get_paths_of_length*, which is given in Figure 6.3 on the facing page: It takes as input a complete type-6 path structure $\Pi = (\mathcal{P}, \mathcal{B}, \mathcal{E})$ with limit $i-1$ and the new limit i; it returns a path set $\mathcal{P}_{res}$, so that $\mathcal{P} \cup \mathcal{P}_{res}$ is the path set of the complete type-6 path structure with limit i. Note that although *get_path_structure_with_limit_1* does not call *get_paths_of_length* with a complete type-6 path structure with limit 0, this borderline case yields a correct result.

The new paths are found in two steps. First, *get_paths_of_length* constructs a set of candidate paths by concatenating all paths in $\mathcal{P}$ of length $i-1$ with all paths in $\mathcal{P}$ of length 1. It then filters out those candidate paths that are not necessary for a complete type-6 path structure, i. e., it rejects irrelevant paths and paths that are composed regarding Π. Hereby, the function *is_necessary*$(P,\ \mathcal{P}_{all},\ \mathcal{B},\ \mathcal{E})$ decides whether or not to accept path P: It returns *true* if a ground instance of P is neither replaceable by a ground instance of a path in $\mathcal{P}_{all}$ nor composed regarding $\mathcal{B}$ and $\mathcal{E}$; it returns *false* otherwise.

Input: i – an integer, the new limit
Π – the complete type-6 path structure with limit $i-1$
Output: $\mathcal{P}_{res}$ – a set of paths of length i

funct *get_paths_of_length*$(i,\ \Pi)$
$(\mathcal{P}, \mathcal{B}, \mathcal{E}) \coloneqq \Pi$
$\mathcal{P}_b \coloneqq \{P \mid P \in \mathcal{P},\ \text{length of } P \text{ is } i-1\}$
$\mathcal{P}_e \coloneqq \{P \mid P \in \mathcal{P},\ \text{length of } P \text{ is } 1\}$
$\mathcal{P}_{cand} \coloneqq \{P_b \circ P_e \mid P_b \in \mathcal{P}_b, P_e \in \mathcal{P}_e\}$

$\mathcal{P}_{res} \coloneqq \emptyset$
while $\mathcal{P}_{cand} \neq \emptyset$ **do**
choose $P \in \mathcal{P}_{cand}$
$\mathcal{P}_{cand} \coloneqq \mathcal{P}_{cand} \setminus \{P\}$
$\mathcal{P}_{all} \coloneqq \mathcal{P} \cup \mathcal{P}_{res} \cup \mathcal{P}_{cand}$
if $(\mathit{is\_necessary}(P,\ \mathcal{P}_{all},\ \mathcal{B},\ \mathcal{E}))$
then $P_{res} \coloneqq P_{res} \cup \{P\}$
fi
od
return $\mathcal{P}_{res}$
.

Figure 6.3: The function *get_paths_of_length* calculates the set of paths that is necessary to extend the limit of a complete type-6 path structure by one.

The relevance of a path depends on its replaceability and on an ordering relation among paths (cf. Definition 5.13 on Page 74). Hereby, t he latter guarantees that if two paths of the same length can replace each other then one of them is added to $\mathcal{P}_{res}$ and the other one is rejected. The ordering relation is determined by the order in which *get_paths_of_length* chooses paths from the set $\mathcal{P}_{cand}$ of candidates: A path P is ordered before a path P', i. e., $P < P'$ if P is chosen after P'. As a consequence, the resulting path set is implementation dependent.

Note that some of the paths that *get_paths_of_length* accepts are composed regarding the final complete type-6 path structure. A call of this function with limit i accepts paths that are minimal regarding Π, which is a path structure with limit $i-1$. As path structures with higher limit can comprise additional starts and stops, some of the paths in $\mathcal{P}_{res}$ can be composed regarding the final path structure. These unnecessary paths are used to construct candidates subsequent calls of *get_paths_of_length*, thus result in further composed paths.

Fortunately, these unnecessary composed paths are harmless: As a path is composed if its prefix is composed, candidates that result from an unnecessary path are rejected as soon as new starts and stops reveal their composedness. Although the unnecessary paths are part of the final path structure, we know from Lemma 5.5 on Page 71 that they do not result in additional starts and stops.

Input: Π – a path structure
$\mathcal{B}_{ext}$ – a set of starts
$\mathcal{E}_{ext}$ – a set of stops
Output: $(\mathcal{B}_{res}, \mathcal{E}_{res})$ – a pair of a set of starts and a set of stops

funct *find_completion*$((\mathcal{P}, \mathcal{B}_{acc}, \mathcal{E}_{acc}), \mathcal{B}_{ext}, \mathcal{E}_{ext})$
  if $\mathcal{B}_{ext} = \emptyset \wedge \mathcal{E}_{ext} = \emptyset$
    then return $(\mathcal{B}_{acc}, \mathcal{E}_{acc})$
    else $(\mathcal{B}_1, \mathcal{E}_1) :=$ *get_be_from_paths*$(\mathcal{P}, \mathcal{B}_{acc}, \mathcal{E}_{ext})$
      $(\mathcal{B}_2, \mathcal{E}_2) :=$ *get_be_from_paths*$(\mathcal{P}, \mathcal{B}_{ext}, \mathcal{E}_{acc})$
      $(\mathcal{B}_3, \mathcal{E}_3) :=$ *get_be_from_paths*$(\mathcal{P}, \mathcal{B}_{ext}, \mathcal{E}_{ext})$
      $\mathcal{B}_{all} := \mathcal{B}_{acc} \cup \mathcal{B}_{ext}$
      $\mathcal{E}_{all} := \mathcal{E}_{acc} \cup \mathcal{E}_{ext}$
      $\mathcal{B}_{new} := (\mathcal{B}_1 \cup \mathcal{B}_2 \cup \mathcal{B}_3) \setminus \mathcal{B}_{all}$
      $\mathcal{E}_{new} := (\mathcal{E}_1 \cup \mathcal{E}_2 \cup \mathcal{E}_3) \setminus \mathcal{E}_{all}$
      $(\mathcal{B}_{res}, \mathcal{E}_{res}) :=$ *find_completion*$((\mathcal{P}, \mathcal{B}_{all}, \mathcal{E}_{all}), \mathcal{B}_{new}, \mathcal{E}_{new})$
      return $(\mathcal{B}_{res}, \mathcal{E}_{res})$
  fi
.

Figure 6.4: The function *find_completion* computes the starts and stops necessary to complete the new type-6 path structure.

6.1.5 Extending the Limit: Additional Starts and Stops

After finding the additional paths, the recursive function *find_completion* finds the starts and stops necessary to complete the new type-6 path structure; it is given in Figure 6.4. The input of this function is a type-6 path structure $(\mathcal{P}, \mathcal{B}_{acc}, \mathcal{E}_{acc})$ with limit i and two sets of facts: a set $\mathcal{B}_{ext}$ of new starts and a set $\mathcal{E}_{ext}$ of new stops; its output is a set of starts $\mathcal{B}_{res}$ and a set of stops $\mathcal{E}_{res}$ such that $(\mathcal{P}, \mathcal{B}_{res}, \mathcal{E}_{res})$ is a complete type-6 path structure with limit i. The new starts and stops are calculated by the function *get_be_from_paths*$(\mathcal{P}, \mathcal{B}, \mathcal{E})$, which finds all starts and stops which result from paths in $\mathcal{P}$ that connect a fact in $\mathcal{B}$ to a fact in $\mathcal{E}$.

A single execution of *find_completion* computes all starts $\mathcal{B}_{new}$ and all stops $\mathcal{E}_{new}$ which result from every path $P \in \mathcal{P}$ that connects a start in $\mathcal{B}_{acc} \cup \mathcal{B}_{ext}$ with a stop in $\mathcal{E}_{acc} \cup \mathcal{E}_{ext}$, excluding those that connect a start in $\mathcal{B}_{acc}$ with a stop in $\mathcal{E}_{acc}$. As $\mathcal{B}_{new}$ and $\mathcal{E}_{new}$ can be connected by paths in $\mathcal{P}$, too, *find_completion* recurses until it cannot find any new starts and stops. Note that $\mathcal{B}_{new}$ comprises only new facts that are not already facts in $\mathcal{B}_{acc}$ and $\mathcal{B}_{ext}$; the same is true for $\mathcal{E}_{new}$. This way, the arguments of calls to *get_be_from_paths* have always disjoint cross products during a recursive descent of *find_completion*.

6.1.6 The Function *get_be_from_paths*

The function *get_be_from_paths*, as given in Figure 6.5 on the facing page, computes all starts and stops that result from a given set $\mathcal{P}$ of paths, a set $\mathcal{B}$ of starts, and a set $\mathcal{E}$ of stops by enumerating every element $(P, f_b, f_e) \in \mathcal{P} \times \mathcal{B} \times \mathcal{E}$. If P connects f_b to f_e then it calls *get_be_from_path*(P) and collects the resulting starts and stops.

Input: $\mathcal{P}$ – a path set
$\mathcal{B}$ – a set of starts
$\mathcal{E}$ – a set of stops
Output: $(\mathcal{B}_{res}, \mathcal{E}_{res})$ – a pair of a set of starts and a set of stops

funct *get_be_from_paths*$(\mathcal{P}, \mathcal{B}, \mathcal{E})$
  $\mathcal{B}_{res} := \emptyset$
  $\mathcal{E}_{res} := \emptyset$
  $\mathcal{P}_{be} := \{P \mid P \in \mathcal{P},\ f_b \in \mathcal{B},\ f_e \in \mathcal{E},\ P \text{ connects } f_b \text{ to } f_e\}$
  foreach $P \in \mathcal{P}_{be}$ **do**
    $(\mathcal{B}_P, \mathcal{E}_P) := get\_be\_from\_path(P)$
    $\mathcal{B}_{res} := \mathcal{B}_{res} \cup \mathcal{B}_P$
    $\mathcal{E}_{res} := \mathcal{E}_{res} \cup \mathcal{E}_P$
  od
  return $(\mathcal{B}_{res}, \mathcal{E}_{res})$
.

Figure 6.5: The function *get_be_from_paths*$(\mathcal{P}, \mathcal{B}, \mathcal{E})$ calculates all starts and stops that result from paths in $\mathcal{P}$ that connect a fact in $\mathcal{B}$ to a fact in $\mathcal{E}$.

Hereby, function *get_be_from_path* computes the set of starts and the set of stops that result from a path according to the definition of type-6 path structures (cf. Page 97). In particular, the result of *get_be_from_path*(P) is the pair of fact sets $(\mathcal{B}_P, \mathcal{E}_P)$, where $\mathcal{B}_P$ is the union of all regular endings of proper sidepaths of P and $\mathcal{F}_\Psi^{through}(P)$; and $\mathcal{E}_P$ is the union of all beginnings of proper sidepaths of P and $\mathcal{F}_\Psi^{through}(P)$.

6.1.7 Soundness and Completeness of Path Reduction

Path reduction is a sound and complete reduction technique. Hereby, soundness means that a solution to the reduced planning problem is a solution to the original planning problem. As path reduction does not change the initial state and the goal of the original planning problem and the reduced action set is a subset of the original one, path reduction is sound.

The completeness of path reduction follows from the sufficiency of type-5 path structures. The complete type-5 path structure, as defined in Section 5.7.4 on Page 96, suffices for its planning problem. Therefore, a reduction strategy is complete if it accepts all actions of paths in the complete type-5 path structure and their corresponding actions in the unstructured part of the planning problem. In the following, we see that this assumption holds, thus *path_reduction* is complete.

By definition, the complete type-6 path structure Π_6 with limit $2^{|\mathcal{F}_\Psi|} - 1$ comprises all paths in a complete type-5 path structure Π_5 for the same problem. Furthermore, Π_5 and Π_6 have the same sets of starts and the same sets of stops: Every path in Π_5 is a path in Π_6 and a path regarding Π_6 at the same time, so that Π_5 has at most the starts and stops of Π_6. Furthermore, every minimal path P in Π_6 that is a path regarding Π_6 is a path in Π_5, too, so that P results only in starts and stops that are in Π_5. As composed

paths in Π_6 do not yield additional starts and stops, Π_5 and Π_6 have the same sets of starts and stops, respectively.

It is left to show that path reduction does indeed compute the complete type-6 path structure with limit $2^{|\mathcal{F}_\Psi|}-1$ for a planning problem: First, we show that if *path_reduction* executes the loop $2^{|\mathcal{F}_\Psi|}-1$ times then the conjecture holds. In the second step, we assure that the two termination criteria $i \leq 2 \cdot max\{s \mid C \in \mathcal{C},\ s \text{ is the size of } C\}$ and $\mathcal{P}_{new} = \emptyset$ do not change the result of *path_reduction*.

We show the first step by induction: According to our assumption, *path_reduction* executes the loop $2^{|\mathcal{F}_\Psi|}-1$ times. The base case is the construction of the complete type-6 path structure with limit 1 by function *get_path_structure_with_limit_1*, which is covered in Section 6.1.3 on Page 99.

Our induction hypothesis is that if $\Pi = (\mathcal{P}, \mathcal{B}, \mathcal{E})$ is the complete type-6 path structure with limit i then $(\mathcal{P}_{next}, \mathcal{B}_{next}, \mathcal{E}_{next})$ is the complete type-6 path structure with limit $i+1$. Set $\mathcal{P}_{new}$ comprises all relevant paths of length $i+1$ that are minimal regarding Π. The call to *get_be_from_paths* computes the starts and stops from paths in $\mathcal{P}_{new}$ that connect a start of Π with a stop of Π. Then $\mathcal{B}_{ext}$ and $\mathcal{E}_{ext}$ are exactly those starts and stops that *additionally* result from the paths in $\mathcal{P}_{new}$. The subsequent call to *find_completion* finds all starts and stops of the complete type-6 path structure with limit $i+1$. Therefore, the induction hypothesis holds and path reduction does indeed compute the complete type-6 path structure with limit $2^{|\mathcal{F}_\Psi|}-1$ if the loop is executed $2^{|\mathcal{F}_\Psi|}-1$ times.

6.1.8 Termination Criteria

The previous section shows that path reduction is complete if the loop of *path_reduction* is executed $2^{|\mathcal{F}_\Psi|}-1$ times. Because of two termination criteria, the loop usually terminates before reaching this limit. We now show that these criteria leave the result of path reduction unchanged.

Termination in case $\mathcal{P}_{new} = \emptyset$: The loop terminates in iteration i if it fails to find any path of length $i+1$ because further iterations always yield the same result. If l is the length of the longest path in a complete type-6 path structure with limit $2^{|\mathcal{F}_\Psi|}-1$ and Π_l is the complete type-6 path structure with limit l for the same planning problem then Π_{l+1} does not comprise any path of length $l+1$ because all paths of this length are either irrelevant or composed: *get_paths_of_length* constructs new candidates of length i by composing accepted paths of length $i-1$, thus it will never construct any candidate of length $l+1$ or longer. Therefore, all complete type-6 path structures with limit l or larger are the same. In other words, terminating the loop in iteration $l+1$ yields the complete type-6 path structure with limit $2^{|\mathcal{F}_\Psi|}-1$.

Termination after iteration $2 \cdot max\{s \mid C \in \mathcal{C},\ s \text{ is the size of } C\}$: Path reduction computes the reduced planning problem from the path structure Π that results from the final iteration of the loop. In the following, we will see that if a path P in Π passes through a fact of its c-invariant three times or more then its exclusion from Π does not change the reduced planning problem. As this is the case for every path through c-invariant C that is of length $2|C|+1$ or longer, we can terminate the loop after a number of iterations that equals twice the size of the largest c-invariant.

If a path P in Π passes through a fact of its c-invariant three times or more then there are two paths P_1 and P_2 in Π that are partial sequences of P and that use all actions of P. In particular, we can write P as $\big(T' \circ T_1 \circ T'' \circ T_2 \circ T''', C\big)$, where T_1 and T_2 are circular in C and begin with the same fact of C. Then $P_1 = \big(T' \circ T_1 \circ T'' \circ T''', C\big)$ and $P_2 = \big(T' \circ T'' \circ T_2 \circ T''', C\big)$ are paths in Π, too, and if P is a path regarding Π then so are P_1 and P_2. Furthermore, P_1 and P_2 use all actions that are in P.

The combined beginnings and endings of sidepaths in P_1 and P_2 subsume the beginnings and endings of sidepaths in P, respectively: Minimal paths in a shortest solution are free (cf. Lemma 5.28 on Page 96), thus P has at most one sidepath with core through any c-invariant. In particular, if P has a proper sidepath Q through C' with core then all path-actions of C' in P are in Q. As a consequence, the first action of Q is the first action of a sidepath either of path P_1, of path P_2, or of both. Likewise, the last action of Q is the last action of a sidepath either of P_1, of P_2, or of both. In other words, the removal of P from Π does not change the result of path reduction. As the same holds for all paths of length $2|C| + 1$ or longer, we can terminate the loop at this bound.

6.1.9 Path Reduction is Systematic

The previous section showed that path reduction is complete and sound; we now see that it is systematic. Hereby, we mean that path reduction never extends a path structure by a path twice and it generates the starts and stops from a path at most once, i. e., it never calls *get_be_from_path* for a path a second time. If a call to a function does not result in such a superfluous extension or call to *get_be_from_path* we say this call is *systematic*.

Path reduction is systematic in respect to path extension simply because the initialization of *path_reduction* finds paths of length one and the i-th iteration of the loop body generates paths of length $i + 1$, thus a path is generated and used for extension at most once.

It takes more effort to show that path reduction is systematic in respect to function *get_be_from_path*. Let us first consider the function *get_be_from_paths*, which takes as arguments a set $\mathcal{P}$ of paths, a set $\mathcal{B}$ of starts, and a set $\mathcal{E}$ of stops. It calls *get_be_from_path* for every element $\big(P, f_b, f_e\big)$ of the cross product $\mathcal{P} \times \mathcal{B} \times \mathcal{E}$ if P connects f_b to f_e. Therefore, *path_reduction* is systematic if it never results in two distinct calls to *get_be_from_paths* whose cross product of the arguments have a common element.

Now we consider a call to the recursive function *find_completion*: If its arguments $\mathcal{B}_{acc}$ and $\mathcal{E}_{ext}$ are disjoint, as well as $\mathcal{E}_{acc}$ and $\mathcal{E}_{ext}$, then these respective arguments will be disjoint in all calls of the subsequent recursive descent. As previously considered starts and stops are collected in the sets $\mathcal{B}_{acc}$ and $\mathcal{E}_{acc}$, respectively, this means that all calls to *get_be_from_paths* have disjoint cross products during the recursive descent. Furthermore, if the facts in $\mathcal{B}_{ext}$ and $\mathcal{E}_{ext}$ of the initial call are new, i. e., if every previous call to *find_completion* used sets $\mathcal{B}'_{ext}$ and $\mathcal{E}'_{ext}$ that were disjoint from $\mathcal{B}_{ext}$ and $\mathcal{E}_{ext}$, respectively, then all calls to *get_be_from_paths* by the current recursive descent are systematic. Therefore, the call *find_completion* by *get_path_structure_with_limit_1* is systematic: It is the first call to this function and $\mathcal{B}_{acc}$ and $\mathcal{E}_{acc}$ are the empty set.

The function *path_reduction* uses *get_be_from_paths* and *find_completion*; we now show that these calls are systematic, too. First note that all calls to *find_completion* by *find_completion* use arguments $\mathcal{B}_{ext}$ and $\mathcal{E}_{ext}$ that are explicitly set to be disjoint from

$\mathcal{B}$ and $\mathcal{E}$, respectively. Furthermore, $\mathcal{B}$ and $\mathcal{E}$ comprise all previously used starts and stops, and all starts and stops in $\mathcal{B}_{ext}$ and $\mathcal{E}_{ext}$ are new, respectively. Therefore, the calls to *get_be_from_paths* in *find_completion* are systematic.

We are left with the call to *get_be_from_paths* in the loop of *path_reduction*. As the path set of this call solely comprises new paths, i. e., paths of the length that equals the new limit, this call is systematic, too.

6.2 General Level Implementation

The previous section presents the algorithm of path reduction for the ground case. As we have noted in the introduction on Page 12, the use of grounded representations has apparent disadvantages, e. g., they can be substantially larger than their corresponding parameterized representation. Furthermore, stating a problem in a parameterized way usually reveals some of its structure and of its underlying world. This information is lost by transforming it to the ground level. To overcome these limitations we present in this section a lifted version of the path reduction algorithm.

6.2.1 Schematic Generalization

The idea of lifting was introduced by Robinson [57] to find a proof procedure for first order logic in analogy to techniques used for propositional logic. Lifting is the transformation of a procedure involving ground facts into a procedure involving variables so that any computation of the first procedure is an instance of the latter. Here, we use the term to indicate that path reduction handles a parameterized planning problem without transforming it into its ground representation.

The standard approach to lift a planning technique to the general level is the use of *parameterized actions*. A parameterized action is a pair $\big(o(\vec{x}), \mathcal{R}\big)$, where $\vec{x}$ is a sequence of pairwise distinct *parameters*, i. e., variables that can be bound to objects, o is an action schema with parameters $\vec{x}$, and $\mathcal{R}$ is a set of constraints[1] on $\vec{x}$. It represents a set of ground actions that results from instantiations of $o(\vec{x})$. In particular, $\big(o(\vec{x}), \mathcal{R}\big)$ represents a ground action o if and only if o results from instantiating $\vec{x}$ in a way that satisfies $\mathcal{R}$.

Recall that we have used action schemata before: Example 3.16 on Page 36 defines the blocks world via three action schemata. In this domain, the parameterized action $\big(\mathit{move\text{-}from}(?x, ?y), \{?x \in t_{block}, ?y \in t_{block}, ?x \neq ?y\}\big)$ represents all ground actions that move a block from the table onto another block. Note that every parameter of the action is bound by a *type constraint*. Hereby, t_{block} is a *type*, i. e., a finite set of objects and $?x \in t_{block}$ is a type constraint that restricts the instantiations of $?x$ to objects of type block. In general, parameters are always bound by type constraints, so that a parameterized action always represents a finite set of ground actions.

We define *parameterized facts*, *parameterized action sequences*, and *parameterized paths* in a similar way: A parameterized fact is a pair $\big(f(\vec{x}), \mathcal{R}\big)$, where $\vec{x}$ is a vector of parameters, f is a predicate schema with parameters $\vec{x}$, and $\mathcal{R}$ is a set of constraints on $\vec{x}$. A

[1] We use $\mathcal{R}$ to denote a set of constraints because $\mathcal{C}$ is already used to denote a set of c-invariants.

parameterized action sequence is a pair $\big(T(\vec{x}), \mathcal{R}\big)$, where $T(\vec{x})$ is a sequence of action schemata $\big\langle o_1(\vec{x}_1), o_2(\vec{x}_2), \ldots \big\rangle$, $\vec{x}$ is the vector of all parameters in $\vec{x}_1, \vec{x}_2, \ldots$, and $\mathcal{R}$ is a set of constraints on $\vec{x}$. Likewise, a parameterized path is a triple $\big(T(\vec{x}), C(\vec{x}), \mathcal{R}\big)$, where $T(\vec{x})$ is a sequence of action schemata, $C(\vec{x})$ is a schema of a c-invariant, and $\mathcal{R}$ is a set of constraints on $\vec{x}$. Hereby, a parameterized path represents the set of ground paths $\big(T, C\big)$ that results from instantiating $\vec{x}$ in all ways that satisfy the constraints in $\mathcal{R}$. From now on, we use the terms *actions*, *fact*, *path*, and *plan* for parameterized actions, facts, paths, and plans respectively. To refer to, e. g., a ground instance of a parameterized path we say *ground path*.

Parameterization lifts a ground planning algorithm to a non-deterministic algorithm that uses a general level representation: If each ground instance of an action, a fact, a path, and a plan is replaced by its parameterized version then an equality test during the execution of the algorithm becomes a non-deterministic operation that either succeeds and results in an equality constraint or fails and results in an inequality constraint. These constraints are added to the corresponding constraint sets and the algorithm continues as long as the constraints remain consistent, i. e., the represented set of ground plans stays non-empty. If the constraint set becomes unsatisfiable, i. e., there is no instantiation of the parameters that results in a ground plan then the plan is rejected and the algorithm backtracks. The lifted algorithm succeeds if it terminates with a satisfiable constraint set, in which case every satisfying ground instance is a solution.

6.2.2 Deciding Path Replaceability by Theorem Proving

Although path reduction uses parameterized actions and paths, the standard approach, as described in the previous section, is not sufficient for a reduction technique based on path replacement: The usual goal of a planner is to synthesize a single ground plan, i. e., a parameterized plan for which there exists (at least) *a single* satisfying assignment. In contrast, the replacement of a parameterized path requires *all* corresponding ground paths to be replaceable. As the standard approach guarantees only the existence of an individual assignment rather than a set of such, path reduction has to use additional techniques.

Our solution is to decide the replacement of a parameterized path by theorem proving. Suppose the function *is_necessary* (cf. Page 101) has to test whether a candidate path can replace a target path. First, it compares the paths to determine if the first is a *replacement candidate*, i. e., if it has a ground instance that can replace a ground instance of the target path. This test of all conditions on path replaceability, as given in Definition 4.11 on Page 53, results in a set of constraints that have to be satisfied in order to yield a valid replacement. In the second step, all constraints resulting from the target path, the replacement candidate, and the comparison are combined to a formula in first order logic that is true if the target path is either composed, replaceable by the candidate, or both. The formula is given to a theorem prover that acts as black box and simply returns true or false, which is then negated and given back as the result of function *is_necessary*.

Dividing the path replaceability check into path comparison and a theorem proving step has several advantages: First of all, there is no need to axiomatize, e. g., path correspondence. All conditions on corresponding paths are tested by the main program and only the resulting constraints are recorded, which allows to keep the resulting formulas

small. Furthermore, failure is detected early, so that the number of unsuccessful proof attempts is reduced. In the following, we describe our implementation of path reduction, followed by a detailed example of how to decide path replaceability by theorem proving.

6.2.3 Implementation Details

Currently, the implementation of path reduction is still in an experimental stage, so that the design of the implementation, the choice of programming language, and the way to connect the proof system is aimed at flexibility and simplicity rather than efficiency. As a consequence, the performance of the current implementation of path reduction is not competitive and its applicability is restricted.

The implementation of path reduction can be described as follows: The input comprises two files, a domain and a problem description, in PDDL [49,5] and DKEL [33] format. An example input is given in Appendix A on Page 125. PDDL, the planning domain definition language, describes a planning problem via action schemata whose parameters are typed and are restricted by equality and inequality constraints. Path reduction creates a parameterized action for each such action description. DKEL, the domain knowledge definition language, is designed as an extension of PDDL to represent parameterized domain knowledge. Path reduction accepts c-invariants in DKEL format as part of the input problem; they can be found automatically by TIM_dkel, a reimplementation of TIM that uses DKEL as output format.[2]

After parsing the input, path reduction sets up the basic blocks of paths and path structures as described in Chapter 5. In particular, it computes the path-actions of the domain, i. e., it partitions the action set into $\mathcal{O}^{\mathcal{C}}$ and $\mathcal{O}^{\overline{\mathcal{C}}}$, and generates the sets $\mathcal{F}^{end}$, $\mathcal{F}^{\mathcal{C}}$, and $\mathcal{F}^{\overline{\mathcal{C}}}$. It initializes the reduction process by constructing the complete type-6 path structure with limit one and incrementally computes complete type-6 path structures with increasing limit, as described in Section 6.1 on Page 97. It then extracts the actions of the complete type-5 path structure from the final path structure of the loop and adds the necessary actions of $\mathcal{O}^{\overline{\mathcal{C}}}$. Path reduction returns the reduced planning domain and the problem either in DKEL or in pure PDDL, i. e., with or without c-invariants.

Path reduction is implemented in Prolog using the ECL$^{\text{i}}$PS$^{\text{e}}$ constraint programming system [68]. Parameters and their type constraints are represented as set variables, i. e., variables that are to be unified with exactly one element out of a finite set of values. Equality and inequality constraints on parameters are applied to these variables incrementally and the constraint solver propagates these constraints to detect unsatisfiable constraints sets early.

Path reduction uses a theorem prover as black box; in other words, it simply constructs the formula, applies the prover, and parses the result. Hereby, the formula is written in DFG syntax [31]. This language is designed as a common input of theorem provers, so that path reduction can use a variety of provers without any knowledge about them. Currently, path reduction uses the theorem prover E [62]. The input to a prover is called a *problem* in the parlance of the DFG syntax. To avoid confusion with planning problems, we call it a *proof problem*.

[2] The source of TIM_dkel is available at `http://thispla.net/arbeit.html`

Unfortunately, these design choices have significant impact on the performance and applicability of path reduction: First of all, the ECL^{i}PS^e system does not allow to extract the current constraint set for a given set of variables. As a consequence, our implementation maintains explicit constraint sets that correspond to the internal sets of the ECL^{i}PS^e constraint solver and the formulas to decide path replaceability are constructed from these explicit constraints. Obviously, this approach is inefficient because we need every constraint set twice. As a consequence, it is not possible to process planning problems with many c-invariants and long paths, for example tyreworld and logistics. In addition, the constraint solver applies sophisticated propagation techniques that make the internal sets more concise than their explicit correspondence. A reimplementation of path reduction in a constraint system that allows to extract constraint sets would improve its performance and applicability significantly.

Furthermore, the black box approach to theorem proving results in an unnecessary overhead: Every time the function *get_paths_of_length* has to decide upon the replaceability of a path, path reduction has to construct a proof problem and to write it to a file. The prover then reads this file, decides the specified conjecture, and writes its output in a stream, which is read and parsed again. An integrated prover could render this communication overhead obsolete.

6.2.4 Deciding Path Replaceability – An Example

Path reduction uses a theorem prover to decide whether the function *get_paths_of_length* accepts or rejects a path (cf. Figure 6.3 on Page 101). For this, it constructs a proof problem that is true if and only if that path is to be accepted. Such a proof problem in DFG syntax is more than simply a conjecture. In particular, it includes declarations and auxiliary axioms. The subsequent sections explain the construction of a proof problem for path replaceability and exemplify its different parts by the following example from the blocks world. Note that from now on the terms *predicate* and *function* refer to a predicate and a function in DFG syntax, respectively.

Example 6.1 (Construction of a Proof Problem)
We exemplify the construction of such a proof problem for a particular combination of a path P, a path set $\mathcal{P}_{all}$, and a type-6 path structure Π in a blocks world domain (cf. Example 3.16 on Page 36) with four blocks. In other words, t_{block} is the set $\{b_0, b_1, b_2, b_3\}$.

Various paths in the blocks world are replaceable. In particular, an indirect move from the table onto a block, i. e., a move that uses intermediate blocks, can always be replaced by a direct move. The following path P is an example of such an indirect move, we call it the target path *of the replacement.*

$$P = \Big(\big\langle \textit{move-from}(?x_1, ?y_1), \textit{move}(?x_1, ?y_1, ?z_1)\big\rangle, C^l(?x_1), \\ \{?x_1 \in t_{block}, ?y_1 \in t_{block}, ?z_1 \in t_{block}, ?x_1 \neq ?y_1, ?x_1 \neq ?z_1, ?y_1 \neq ?z_1\}\Big)$$

Hereby, $C^l(?x)$ denotes the parameterized blocks world c-invariant that corresponds to possible locations for block $?x$. For a specific block $?x$, e. g., for the block b_0, $C^l(b_0)$ denotes the c-invariant $\{\textit{on\_table}(b_0), \textit{on}(b_0, b_1), \ldots, \textit{on}(b_0, b_4)\}$. ■

6.2.4.1 Objects, Types, and Quantification

The conjecture of a proof problem is a formula in first order logic with domain-restricted quantifiers. That is, for each quantified variable we specify a range of constants over which it can take its values; simply said, we restrict quantifiers by types. More precisely, we represent an object of the planning problem as function with arity zero and provide corresponding unique names and closure axioms. Types in planning are sets of objects; in proof problems, they are represented by finite sorts. We restrict quantifiers by annotating their variables with such sorts.

The axioms and declarations of objects and types are the same for all proof problems that are constructed during application of path reduction to a planning problem. In our particular example, they comprise the upper halve of the proof problem, as shown in Figure 6.4 on Page 118.

6.2.4.2 Testing Path Minimality

Whether or not a path is minimal depends on the starts and stops of a specific path structure. In particular, Definition 5.11 on Page 71 states that a path is composed if its longest proper infix passes through a start and a stop of that path structure and the start is either before or simultaneously with the stop. Consequently, some instances of a parameterized path can be composed while the others are minimal.

We encode the starts and stops by two classes of predicates and their corresponding auxiliary axioms: For each fact f we define the predicate f_in_b and construct an auxiliary axiom $\forall\vec{x}.\ f\_in\_b(\vec{x}) \leftrightarrow$ "$f(\vec{x})$ is a start" that encodes f as start, i. e., that sets $f\_in\_b(\vec{x})$ to true for an instance of $f(\vec{x})$ if and only if that instance is a start. Likewise, we define f_in_e for f as stop and construct an auxiliary axiom for $f\_in\_e(\vec{x})$ that sets an instance of $f(\vec{x})$ to true if and only if that instance is a stop. Example 6.2 presents two such auxiliary axioms. The additional predicates for starts and stops allow to encode path composition by the formula

$$F^{comp}(\vec{x}) =_{def} \bigvee_{2\leq i\leq n-1} \left(f^i\_in\_b(\vec{x}_i) \wedge \left(\bigvee_{i\leq j\leq n-1} f^j\_in\_e(\vec{x}_j) \right) \right)$$

for a path with parameter vector $\vec{x}$ that passes through the facts $f^1(\vec{x}_1), f^2(\vec{x}_2), \ldots, f^n(\vec{x}_n)$. Note that the direct encoding of starts and stops as part of the conjecture of the proof problem would often yield an unwieldy formula: Firstly, a path can pass through the same fact twice or more. Furthermore, for paths of length four and longer, the formula F^{comp} uses the information about stops repetitively. Auxiliary axioms abbreviate starts and stops, thus evade these unnecessary duplications.

We illustrate the auxiliary axioms and the formula F^{comp} by our example from the blocks world.

Example 6.2 (Construction of a Proof Problem, continued)
In our example, the sets of starts and stops are given as follows.

$$\mathcal{B} = \left\{ \left(on(?x_3, ?y_3), \{ ?x_3 \in t_{block}, ?y_3 \in t_{block}, ?x_3 \neq b_4, ?y_3 \neq b_0, ?x_3 \neq ?y_3 \} \right) \right\}$$

$$\mathcal{E} = \left\{ \left(on\_table(?x_4), \{ ?x_4 \in t_{block} \} \right) \right\}$$

These sets result in the two auxiliary axioms

$$\forall x \in t_{block}, y \in t_{block}.\ on\_in\_b(x, y) \leftrightarrow x \neq b_4 \wedge y \neq b_0 \wedge x \neq y$$

$$\forall x \in t_{block}.\ on\_table\_in\_b(x)$$

of the proof problem, compare the end of part `list_of_formulae(axioms)` *on Page 118. Further auxiliary axioms set all instances of all other predicates for starts and stops to false.*

The target path P passes through the three facts on-table(?x_1), on(?x_1, ?y_1), and on(?x_1, ?z_1), so that the path is composed if and only if on(?x_1, ?y_1) is a start and a stop. Hence, our example conjecture encodes path composition by the formula

$$F^{comp}(x_1, y_1, z_1) =_{def}\ on\_in\_b(x_1, y_1) \wedge on\_in\_e(x_1, y_1).$$ ■

6.2.4.3 Testing Path Replaceability

The definition of path replacement (cf. Page 53) gives several conditions for the valid replacement of a ground path by another. The use of path replacement by path reduction extends this notion in two directions. (1) Path reduction considers parameterized paths. Hereby, a parameterized path is replaceable if each of its ground instances is replaceable. (2) A target path is tested for replaceability against several candidate paths simultaneously, which allows the replacement of each ground instance by a different candidate. This section explains how to encode these extensions into a proof problem.

The first step of testing path replaceability is a comparison between the target path and each candidate. Each such comparison results in sets of constraints that encode the replaceability of the target path by the particular candidate. In other words, every instance of the target path that satisfies the constraints of a set is replaceable by an instance of the candidate. Hereby, the constraints result from the conditions of path replaceability and from the candidate path. For example, the condition that the beginning and ending of corresponding ground sidepaths are the same results in equality constraints that guarantee that the beginnings of the sidepaths are always instantiated to the same ground fact. The other conditions on corresponding paths, e. g., those involving facts in $\mathcal{F}^{end}$ and $\mathcal{F}^{\overline{C}}$, result in similar constraints. As only a valid ground instance of the candidate path can replace the target path, the set also comprises the constraints of the candidate.

The comparison between a target path and a candidate path can result in several sets of constraints. For example, if a candidate is comparable to a target path under two distinct segmentations then path reduction creates two sets of constraints, one for each segmentation; the same is true if there is a choice between different mappings of sidepaths. Path comparison enumerates all possibilities to replace a target path by a candidate and generates the corresponding constraints.

Each constraint set that results from path reduction uses new variables to simplify the construction of the proof problem. As for every ground instance of the target path there has to exist a ground instance of a candidate that replaces it, we bind these variables by existential quantification. Equality constraints between the parameters of the target

path and the candidate guarantee that they are instantiated to the same object, if necessary. Note that the candidate can have more parameters than the target path. Such additional variables are the result of additional added effects of the candidate that are facts in the unstructured part of the domain (cf. Section 4.10 on Page 52 and Point 3 of Definition 3.15 on Page 37).

For a single constraint set, path replaceability is encoded by the formula

$$F^{cand}(\vec{x}) =_{def} \exists \vec{y}.\ F^{eq}(\vec{x}, \vec{y}) \wedge F^{corr}(\vec{y}).$$

Hereby, $\vec{x}$ is the variable list of the target path, $\vec{y}$ is the variable list of the candidate, and F^{eq} comprises equality constraints between the parameters of the target path and the candidate. Furthermore, F^{corr} encodes the constraints that result from the comparison and from the candidate path. Again, we exemplify this formula in our Example 6.1 on Page 109.

Example 6.3 (Construction of a Proof Problem, continued)
Suppose the set $\mathcal{P}_{all}$ of function get_path_of_length *(cf. Figure 6.3 on Page 101) comprises the sole path*

$$P' = \Big(\big\langle \textit{move-from}(?x_2, ?y_2)\big\rangle, C^l(?x_2), \{\, ?x_2 \in t_{block}, ?y_2 \in t_{block}, ?x_2 \neq ?y_2\}\Big).$$

The comparison of paths P and P' yields a single constraint set because they are comparable under the sole segmentation $\big(\langle\,\rangle, \langle \textit{move-from}(?x_2, ?y_2)\rangle\big)$. If a ground instance of P' can replace a ground instance of P then they begin with the same fact, which is expressed by the constraints $?x_1 = ?x_2$ and $?z_1 = ?y_2$. As the remaining conditions do not yield additional constraints, the resulting formulas are

$$F^{eq}(x_1, y_1, z_1, x_2, y_2) =_{def} x_1 = x_2 \wedge z_1 = y_2 \quad \textit{and}$$
$$F^{corr}(x_2, y_2) =_{def} x_2 \neq y_2,$$

thus the combined formula is

$$\begin{aligned} F^{cand}(x_1, y_1, z_1) =_{def}\ & \exists x_2, y_2.\ F^{eq}(x_1, y_1, z_1, x_2, y_2) \wedge F^{corr}(x_2, y_2) \\ =\ & \exists x_2, y_2.\ x_1 = x_2 \wedge z_1 = y_2 \wedge x_2 \neq y_2. \end{aligned}$$

■

6.2.4.4 Combining the Pieces

Function *is_necessary* (cf. Figure 6.3 on Page 101) returns false if each ground instance of a target path P is either composed, replaceable by at least one ground instance of a candidate, or both. The corresponding conjecture, which combines the parts that are developed in the preceding sections, is constructed as follows. Note that this formula actually encodes the inverse of *is_necessary*, i. e., *is_necessary* returns true if and only if the formula is false.

$$\forall \vec{x}.\ F^{path}(\vec{x}) \rightarrow \big(F^{comp}(\vec{x}) \vee F_1^{cand}(\vec{x}) \vee \cdots \vee F_n^{cand}(\vec{x})\big)$$

Example 6.5 (Construction of a Proof Problem, continued)
The combined conjecture of our example is

$$\begin{aligned}\forall x_1, y_1, z_1.\ x_1 \neq y_1 \wedge x_1 \neq z_1 \wedge y_1 \neq z_1 \rightarrow \\ on\_in\_b(x_1, y_1) \wedge on\_in\_e(x_1, y_1) \vee \\ \exists x_2, y_2.\ x_1 = x_2 \wedge z_1 = y_2 \wedge x_2 \neq y_2.\end{aligned}$$

The proof problem in Figure 6.4 on Page 118 shows this formula in the part `list_of_formulae(conjecture)`*. Note that for reasons of brevity, this proof problem does not comprise the list descriptions and the axioms for facts in* $\mathcal{F}^{end}$*. As no action in the blocks world endangers or destroys a fact, the latter are not necessary for the example conjecture.* ■

A proof problem for path replaceability has two further parts that have not been mentioned yet: Firstly, it comprises predicates that result from facts in $\mathcal{F}^{end}$, together with their corresponding auxiliary axioms. These axioms are constructed and used similarly to the auxiliary axioms for path composition. They encode whether or not a path passes through an endangered fact; path comparison uses them to guarantee that the candidate is as safe as the target path in respect to fact endangerment (cf. Section 4.5 on Page 50). Finally, a proof problem in DFG syntax has a mandatory list of descriptions.

6.3 Experiments and Results

The last two sections introduced the algorithm of path reduction and explained its general level implementation. We demonstrate the effects of path reduction by applying it to several example problems and by comparing the results with those of two other reduction techniques. In short, although path reduction shows promising results in some planning domains, it is not suitable as general reduction technique in its current state.

6.3.1 The Loop Domain

Our first example domain is called the loop domain; its definition in DKEL is given Appendix A.1 on Page 125. The loop domain consists of a two-dimensional grid, where each node of the grid is represented by a ground fact. The set of all nodes is a c-invariant and we call the active fact of this c-invariant the *active node*. Actions allow to *move* the active node along one of the two dimensions. In other words, each action allows to change one of the two coordinates of the active node, so that a move of the active node to a node that differs in both dimensions requires at least two actions.

The move of the active node is restricted. The grid is subdivided into four segments by splitting each dimension into two equal halves (cf. Figure 6.6 on the next page). Changing segments is allowed counterclockwise only, so that revisiting a segment requires to pass through the other three segments. Within a segment, the move of the active node is unrestricted. Consequently, the connection of two arbitrary nodes requires at most three actions.

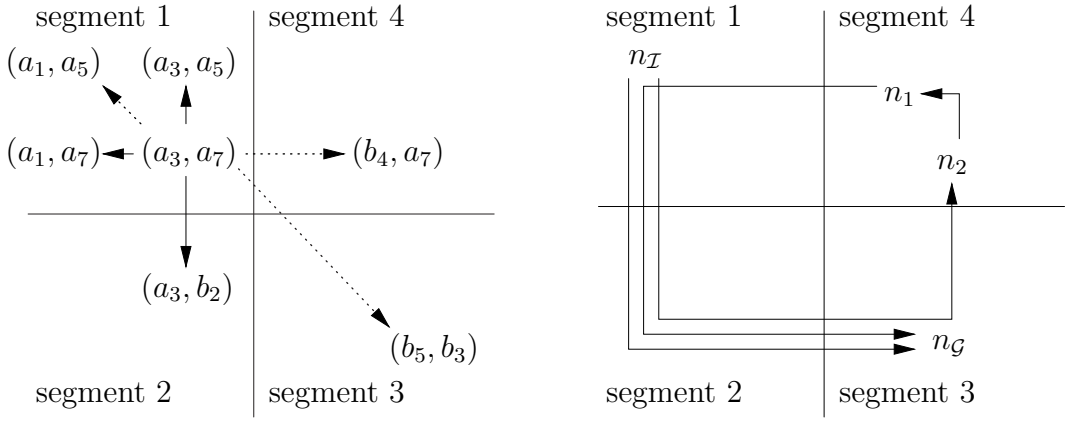


Figure 6.6: The loop domain. The left figure depicts nodes by their coordinates. The node (a_3, a_7) is the active node; solid arrows depict its possible moves by a single action, dotted arrows depict moves that require two actions or more. The right figure shows the nodes that define the grid problems. The initial state specifies $n_{\mathcal{I}}$ as the active node and $n_{\mathcal{G}}$ has to be active in the goal state. During the execution of a solution, the active node has to pass through nodes n_1 and n_2. Here, the arrows depict the paths in the final complete type-6 path structure Π computed by *path_reduction* that connect a start of Π to a stop of Π.

Problems in the loop domain require to move the active node from an initial position to a goal node. Furthermore, the active node has to pass through various other nodes on its way. Thus, a typical solution consists of one circle of the active node through all four segments, followed by some further moves to reach the final segment.

The loop domain poses difficulties for planning systems: Firstly, the size of the ground representation depends cubicly on the side length of the grid, which poses a problem for planning systems based on grounding. Secondly, the structure of the grid prevents some techniques from working. As demonstration, we apply the following planning techniques on loop problems: FF [34], MIPS [15], IPP [42] (without RIFO), and LPG [28] are planners based on grounding. RIFO [50] and RedOp [32] are reduction techniques based on grounding.

We define two loop problems of different size to demonstrate these difficulties: A small instance with grid size 4×4 and a large instance with grid size 15×15. The small instance has 16 nodes that are connected by 640 ground actions. The initial and the goal node is in segment one and three, respectively. Furthermore, the active node has to pass through two nodes in segment four. The large problem differs from the small one only in its size. Its grid has 225 nodes that are connected by 42300 ground actions. The definition of these problems can be found in Appendix A.2 on Page 127.

The application of various planning systems on these problems yields the following results: The planner FF solves the large problem in less than two seconds.[3] and MIPS needs about two minutes. The systems LPG, RIFO, and RedOp cannot process this problem because it exceeds the available resources. IPP claims that it is unsolvable.

All systems can process the small problem. FF and MIPS solve it very quickly. Again, IPP claims that this problem is unsolvable. In other words, its the structure of the grid

[3] All experiments are performed on the same Linux PC with 700 MHz and 192 MB RAM.

that prevents the application of IPP, not the size of the instance. RIFO is able to process the small problem but its application does not yield a reduction; RedOp reduces the number of its ground actions by 85%, which increases the length of the shortest solution from 9 to 19.

The application of path reduction yields a strong decrease in problem size. The final path structure of type-5 comprises four paths, which are depicted on the right in Figure 6.6 on the preceding page. Three of them connect nodes of segment 1 to 3, 1 to 4, and 4 to 3. The fourth path connects two nodes in an arbitrary segment. The reduced planning domain comprises 20 action schemata, compared to 8 schemata of the original domain; it is given in Appendix A.3 on Page 127. The ground instantiation of the reduced problem comprises 200 actions. Two of the original action schemata are completely removed, the others are partially instantiated. For example, the two original schemata that move the active node to another in the same segment are reduced to four ground actions each. The application of path reduction on this problem takes about 115 seconds, three fifth hereof are due to the theorem prover.

Not surprisingly, the reduced version of the large problem is easier to solve and it can be processed by all tested planning systems. Each of FF, MIPS, and LPG find a solution in less than 0.1 seconds. IPP, which initially claimed that the original problem has no solution, solves the reduced problem in less than 0.2 seconds. RIFO and RedOp are able to reduce the problem even further, as a result of the small size and the changed structure of the reduced problem.

In summary, path reduction allows to reduce problems of the loop domain in a way that is not obtainable by RIFO and RedOp. Furthermore, grounding based planning systems can profit from a non-ground reduction technique. The next two sections show these effects more clearly.

6.3.2 Exploiting the Segmentation of Paths

Path reduction identifies action as irrelevant that cannot be identified by RIFO and by RedOp; we demonstrate such a case with the planning domain given in Figure 6.7 on the next page. Hereby, the identification of the replaceable path requires a complex partitioning of the candidate. We define the domain as introduced in Example 3.13 on Page 31. For example, the column that is labeled with $o_1(?x, ?y, ?z)$ defines this action as $\Big(\big(\{f_1(x)\}, \{f_2(?x)\}, \{f_1(?x), q(?z), q(?z)\}\big), \big\{?y \neq ?z\big\}\Big)$. The given domain has the following two comparable paths.

$$P_1 = \Big(\big\langle o_1(?x, ?y, ?z), o_2(?x)\big\rangle, C(?x), \{?x \in t_{obj}, ?y \in t_{obj}, ?z \in t_{obj}, ?y \neq ?z\}\Big)$$
$$P_2 = \Big(\big\langle o_3(?x), o_4(?x)\big\rangle, C(?x), \{?x \in t_{obj}\}\Big)$$

Both paths delete facts of the unstructured part of the domain. Action o_3 of path P_2 deletes a fact of q that depends on the path: If the ground instance of path P_2 begins with, e. g., $f_1(ob_1)$ then o_3 deletes $q(ob_1)$. Action o_1 of path P_1 deletes two instances of facts of q, too. In contrast to the one of o_3, these instances are arbitrary. In other words, the instantiations of the facts q are independent of the beginning of P_1. As a consequence, path

	$o_1(?x, ?y, ?z)$	$o_2(?x)$	$o_3(?x)$	$o_4(?x)$
$C(?x) =$ $\{f_1(?x), f_2(?x), f_3(?x), f_4(?x)\}$	$f_1(?x) \rightarrow f_2(?x)$	$f_2(?x) \rightarrow f_4(?x)$	$f_1(?x) \rightarrow f_3(?x)$	$f_3(?x) \rightarrow f_4(?x)$
Constraints	$?y \neq ?z$			
facts in $\mathcal{F}^{\overline{C}}$	del: $q(?y)$ del: $q(?z)$	add: $q(?x)$	del: $q(?x)$	add: $q(?x)$

Figure 6.7: The Planning Domain Used in Section 6.3.2.

P_2 cannot replace path P_1 under the segmentation $\big(\langle o_3(?x)\rangle, \langle o_4(?x)\rangle\big)$: The definition of replaceability requires that a replacing segment can replace its corresponding action.

Nevertheless, the replacement is possible under the segmentation $\big(\langle\,\rangle, \langle o_3(?x), o_4(?x)\rangle\big)$, despite the arbitrary deleted facts of path P_1: The added effect $p(?x)$ of o_4 cancels out the deleted effect of o_3. In particular, the sequences $\langle o_3(?x), o_4(?x)\rangle$ and the action $o_2(?x)$ do not delete any facts, if limited to $\mathcal{F}^{\overline{C}}_{\Psi}$, and they have the same postcondition:

$$post\Big(\text{limit}\big(o_2(?x), \mathcal{F}^{\overline{C}}_{\Psi}\big)\Big) = post\Big(\text{limit}\big(\langle o_3(?x), o_4(?x)\rangle, \mathcal{F}^{\overline{C}}_{\Psi}\big)\Big) = \big\{q(?x)\big\}$$

Note that P_1 cannot replace P_2 because o_1 deletes two instances of fact q.

Now, suppose a planning problem with an instance of fact f_1 is in the initial state and the corresponding instance of fact f_4 is the goal. Here, the application of RedOp does not eliminate any actions because it is restricted to the replacement of single actions. As neither o_1 nor o_2 is replaceable by an action sequence, it accepts all actions of this domain. The application of RIFO does not eliminate the irrelevant actions either: RIFO chooses between sequences that differ in their use of initial facts and objects; it disregards deleted effects of actions. Consequently, RIFO cannot distinguish between the sequences $\big\langle o_1, o_2\big\rangle$ and $\big\langle o_3, o_4\big\rangle$, so that it accepts them both. Path reduction actually compares the two sequences, i. e., paths, thus it is able to detect and remove the irrelevant one.

6.3.3 Size Matters – The Impact of Grounding

The following domain, we call it the cube domain, shows the effect of path reduction on the size of the propositional representation more clearly. The cube domain consists of a three dimensional grid. Similarly to the loop domain, each node of the grid is represented by a ground fact and the set of nodes is a c-invariant. We call the active fact of this c-invariant the *active node*. Actions allow to *move* the active node along one of the three dimensions. In other words, each action allows to change one of the three coordinates of the active node, so that a move of the active node to a node that differs in all dimensions requires at least three actions. If, for example, $\big\{f(ob_0, ob_0, ob_0)\big\}$ is the initial state and $\big\{f(ob_1, ob_2, ob_3)\big\}$ the goal state of a planning problem then $\big\langle o_1(ob_0, ob_1, ob_0, ob_0), o_2(ob_1, ob_0, ob_2, ob_0), o_3(ob_1, ob_2, ob_0, ob_3)\big\rangle$ is one possible solution of length three.

	$o_1(?x_1, ?x_2, ?y, ?z)$	$o_2(?x, ?y_1, ?y_2, ?z)$	$o_3(?x, ?y, ?z_1, ?z_2)$
$C(?x, ?y, ?z) =$ $\{f(?x, ?y, ?z)\}$	$f(?x_1, ?y, ?z) \rightarrow f(?x_2, ?y, ?z)$	$f(?x, ?y_1, ?z) \rightarrow f(?x, ?y_2, ?z)$	$f(?x, ?y, ?z_1) \rightarrow f(?x, ?y, ?z_2)$

In principle, planners such as FF, LPG, and MIPS can solve cube problems quickly. But a planning problem of this domain with n objects $ob_1, \ldots, ob_n$ comprises $3n^4 - 3n^3$ ground actions and n^3 ground facts. Therefore, if the problem comprises many objects, e. g., more than 25 then they fail. The same is true for RedOp: It takes RedOp 10 minutes to process a cube problem with five objects.

Not surprisingly, parameterized planning techniques scales better in this domain: For example, VHPOP [70], a planner based on parameterized actions, solves cube problems nearly instantly regardless of their size. Likewise, path reduction is less affected by problem size; it requires 6.8, 24.5, and 42.8 seconds to process instances with 5, 25, and 40 objects, respectively.

6.3.4 Path Reduction and Commonly Used Test Domains

We have seen in several examples, e. g., Example 4.1 on Page 39 that many paths in the blocks world are replaceable. Nevertheless, the application of path reduction on blocks world problems does not yield a reduction. The same is true for other commonly used planning domains, as tyreworld and logistics. This lack of reduction strength is a result of our aim of a complete and optimal planning technique. In general, the identification of even a single irrelevant action that is irrelevant for finding a solution or, even more, irrelevant for finding a shortest solution is as hard as planning itself (cf. [50]).

The provisions of path reduction to preserve completeness have the consequence that complete type-5 path structures comprise paths that go "backwards" in time: We expect path structures to comprise paths that are in shortest solutions. In other words, if P is a path in a path structure Π then we expect the existence of a shortest solution S, so that P is a path in S. In this case, sequence S passes through the beginning of P first and then through the ending of P. Unfortunately, path structures also comprise paths for which the beginning is *after* their ending in all shortest solutions, we call them *backwards paths*. Clearly, backwards paths are unnecessary.

Backwards paths result from the construction mechanism of complete path structures, which states that the beginning of a sidepath is a stop and the ending is a start. Consequently, path reduction tries to connect the ending of a sidepath to its beginning. Let us give an example in the blocks world (cf. Example 3.16 on Page 36). A path in the blocks world that connects $on(A, B)$ to $clear(B)$ through the c-invariant C^c_B has a sidepath through c-invariant C^l_A. The beginning of this sidepath is $on(A, B)$ and its ending is, e. g., $on(A, C)$, where C is an arbitrary block that differs from A and B. Therefore, if the initial path connects a start to a stop then path reduction will subsequently connect $on(A, C)$ to $on(A, B)$.

We can reduce the number of backwards paths by an exploitation of overlapping c-invariants, which allows to identify beginnings and endings of sidepaths that do not have to be considered as stops and starts, respectively. In the former example, C^l_A and C^c_B are overlapped c-invariants with the common fact $on(A, B)$. This fact is a start of C^c_B, so it is also a start of C^l_A. Then there is no reason to consider $on(A, C)$ as a start of C^l_A.

Unfortunately, this improvement is not strong enough to make path reduction useful in the blocks world. The considered path has another sidepath through c-invariant C^c_C, which corresponds to the coverage of block C. This c-invariant does not overlap with C^c_B, thus it still results in backwards paths. Therefore, a competitive version of path reduction has to identify and reject backwards paths.

```
begin_problem(path_replacement).

list_of_symbols.
functions[0,1,2,3,4].
predicates[(clear_in_b,1),(on_table_in_b,1),(on_in_b,2),
           (clear_in_e,1),(on_table_in_e,1),(on_in_e,2)].
sorts[t1].
end_of_list.

list_of_formulae(axioms).
formula(not(equal(0,1))).  formula(not(equal(0,2))).
formula(not(equal(0,3))).  formula(not(equal(0,4))).
formula(not(equal(1,2))).  formula(not(equal(1,3))).
formula(not(equal(1,4))).  formula(not(equal(2,3))).
formula(not(equal(2,4))).  formula(not(equal(3,4))).

formula(forall([t1(v1)],or(equal(v1,0),equal(v1,1),equal(v1,2),
                           equal(v1,3),equal(v1,4)))).

formula(t1(0)).  formula(t1(1)).
formula(t1(2)).  formula(t1(3)).
formula(t1(4)).

formula(forall([t1(v1)],equiv(clear_in_b(v1),false))).
formula(forall([t1(v1)],equiv(on_table_in_b(v1),false))).
formula(forall([t1(v2),t1(v1)],equiv(on_in_b(v1,v2),
                        and(and(not(equal(v1,4))),and(not(equal(v2,0))),
                            not(equal(v1,v2)))))).
formula(forall([t1(v1)],equiv(clear_in_e(v1),false))).
formula(forall([t1(v1)],equiv(on_table_in_e(v1),true))).
formula(forall([t1(v2),t1(v1)],equiv(on_in_e(v1,v2),false))).
end_of_list.

list_of_formulae(conjectures).
formula(forall([t1(v3),t1(v2),t1(v1)],
  implies(and(not(equal(v3,v2)),not(equal(v3,v1)),not(equal(v2,v1))),
          or(and(on_in_b(v2,v3),on_in_e(v2,v3)),
             exists([t1(v6),t1(v5)],and(equal(v2,v6),equal(v1,v5)),
                                        not(equal(v6,v5))))))).
end_of_list.
end_problem.
```

Example 6.4: A proof problem in DFG syntax. It is the actual output of path reduction for the problem of Example 6.1 on Page 109, slightly edited for readability and brevity.

Chapter 7

Related Work and Conclusions

In this chapter, we list work that is related to the topics covered in this thesis. We then highlight the main contributions of our research, discuss our results, and give directions for future research.

7.1 Related Work

Planners are problem solving systems that are specialized on planning problems and, not surprisingly, they build on structural features that are commonly present in these problems from the very beginning of the field. The separation between generation and use of domain knowledge, i. e., the use of domain analysis appeared later; an early example is the system Reflect [12] by Dawson and Siklóssy. The success of Graphplan [9], a planner by Blum and Furst which exploited the common structure of planning problems in a novel way, spurred wide interest in domain analysis techniques. Today it is fair to say that all current planning techniques use domain analysis techniques to some degree.

7.1.1 State Invariants

A state invariant is a property that holds in every state of a planning problem. Reasons for the existence of state invariants are *persistence* and *action localness*, i. e., properties that result from the fact that actions change only a small part of a state. Zhang and Foo [71] discuss this connection in general. They also describe the construction of formulas on states whose truth persists regarding the execution of actions, i. e., state invariants. Foo *et. al.* [17] cite an additional reason for the connection between state invariants and action localness: the existence of common conventions in defining planning problems.

Many preprocessing techniques are based on action localness. TIM [18], a system by Fox and Long, examines predicate schemata that occur simultaneously in the preconditions and effects of an action. Edelkamp and Helmert exploit *balanced predicates* in the planner MIPS [14]. Hereby, a predicate is balanced if the number of its instantiations in states is constant regarding the application of actions. The presented c-invariants and ad-invariants are based on action localness, too. Their generation and use is demonstrated by the planner ProbaPla [60] of Scholz.

Another approach to find state invariants automatically is by *generation and test*, where a first phase of the algorithm generates candidate invariants and a second phase rejects or refines those candidates which are not invariant with respect to action application. It allows to find state invariants that have few syntactic restrictions, as demonstrated by Rintanen [56]. Gerevini and Schubert combine the generation and test approach with the use of action localness in their system DISCOPLAN [24,25,26]: First they generate candidate invariants by exploiting action localness. In a second step, they find excuses, i. e., conditions that have to be true in the initial state of a problem. These excuses exclude special cases for which the candidates are not invariant with respect to action application.

State invariants are used by the system Reflect. Kautz and Selman use them in their planner SatPlan [37] to generate additional clauses in their translation from planning to SAT. The planner GRT [53,54] of Refanidis and Vlahavas uses XOR-constraints, a class of state invariants, to decompose planning problems and to complete the goal state of a planning problem. They also have a notion of overlapping invariants that is related to the one introduced for c-invariants (cf. Section 3.1.5 on Page 27). The planner Stan [43] by Fox and Long uses the invariants generated by TIM in various ways, e. g., to build efficient data structures [19] and as a basis for the application of specialized algorithms [20,44,45].

Despite the various ways to find and represent state invariants, many of them are related. Haslum and Scholz propose the language DKEL [33] as a first step towards a unified representation of domain knowledge. In particular, they exemplify a unified way to represent c-invariants and XOR-constraints, as well as the state invariants that are produced by TIM and by DISCOPLAN.

7.1.2 Replacement of Action Sequences and of Paths

The idea of excluding replaceable sequences of actions from the search is used in various planning techniques. Again, an early example for such a system is Reflect. By now, such techniques are mostly concerned with continuous action sequences and plan prefixes, in contrast to non-continuous sequences, i. e., paths.

The reduction technique RedOp [32] by Haslum and Jonsson finds actions that are replaceable by a sequence of other actions, thus the first one can be removed from the domain. RedOp supports two different outputs: Either it reduces the planning problem right away, which results in another planning problem whose solution is a solution to the first. In this case, any planner can use RedOp as preprocessing step. Otherwise, it uses the replaceability information to augment the problem by statements in DKEL.

RSA [61], a technique by Scholz, finds sequences of length two that are replaceable by a single action. He also discusses aspects of sequence replacement in a parallel planning framework. ProbaPla [60], a SAT based planner, uses such sequences to create additional clauses. The section on replaceable sequences of actions is an extension of this work (cf. Section 3.2 on Page 36).

A common use of replaceability is *commutativity pruning*, i. e., pruning from search all but one permutations of a sequence of commutative actions. For example, Godefroid and Kabanza [29] use this notion of replaceability to exclude permutations of independent actions. Hereby, commutativity pruning exhibits a similarity to parallel, total-ordered planning, as demonstrated in SatPlan or Graphplan.

Korf [65] uses commutative pruning for eliminating duplicate nodes in the search space, i. e., he rejects all but one path to a particular node. Hereby, he uses a generate and test approach, which differs from RSA in that it does not calculate replaceable paths to nodes explicitly but memorizes taken paths to reject the others. Note that here the term *path* refers to a sequence of steps in a search space rather than to a path in a c-invariant.

Another example of commutativity pruning is presented by Refanidis and Vlahavas [52], which search the space of plans in a forward manner. When they decide upon adding a new action, they compute a set of prohibited actions that are not applied.

Replaceability information is used by planners based on the *planning by rewriting* paradigm, as introduced by Ambite and Knoblock [2,3]. They use a more elaborate notion of replaceability and hand-coded knowledge to build a neighborhood on plans, which they use to find solutions via local search. Together with Minton [4] they developed a technique to learn replaceable action sequences from example solutions.

7.1.3 Path Reduction

Path reduction is a preprocessing technique that, if given a planning problem, constructs a path set whose actions allow to form a shortest solution to the problem. Consequently, path reduction identifies the remaining actions as *irrelevant*, i. e., negligible.

The prevention of irrelevant actions is a common theme in planning, as planners want to concentrate on relevant and applicable actions. Usually, the treatment of irrelevance is intertwined with the search for a plan. Explicit preprocessing techniques include the analysis of relevance and reachability of facts by Brafman [10]. TIM [18] restricts the possible instantiations of actions by inferring types, thus eliminating irrelevant instantiations, i. e., ground actions. A technique of Gerevini and Schubert [26] has a similar effect. Koehler and Hoffmann [40] use inertia information to prevent irrelevant actions from being instantiated, thus avoiding the need to remove them later on.

A common property of the preceding approaches is that they keep an action if it is used by a solution. In contrast, path reduction eliminates actions that can be part of solutions. RIFO [50] and RedOp [32] are reduction techniques that are similar to path reduction in this respect. They differ from path reduction in that they are independent of additional structural features of planning problems, while path reduction is only applicable to planning problems for which c-invariants are known.

RIFO by Nebel, Dimopoulos, and Koehler identifies irrelevance by building a so-called *fact generation tree*, which is a crude approximation of all solutions to a given problem, backwards from the goals. It then uses one of several strategies to select relevant paths in this tree, hence rejecting facts and actions that are not used by any selected path. All these strategies are incomplete and do not preserve optimality. RIFO gains theoretical completeness by applying the strategies in the order of decreasing reduction strength until one of them succeeds. If all of them fail then it resorts to the original, unreduced problem. RIFO is part of IPP [39], a planner by Koehler *et. al.*, and is also used by the planner FF [35] of Hoffmann.

RedOp by Haslum and Jonsson identifies actions as irrelevant that can be replaced by a conflict-free sequence of other actions; we have mentioned it in the preceding section. Its application preserves completeness but does not preserve optimal solutions.

Bacchus and Teh [6] propose a notion of dynamic relevance that identifies irrelevant partial sequences of a given conflict-free action sequence. They partition the input sequence into two partial sequences, a relevant and an irrelevant one, so that the relevant of the two is conflict-free and can replace the input sequence.

The idea of path reduction is the decomposition of planning problems into subproblems by state invariants on the one hand and the composition of paths, i. e., solutions to these subproblems to a solution of the overall problem on the other. The planner GRT [53,54] decomposes planning problems into subproblems according to XOR-constraints, which are a class of state invariants. It uses the relation between solutions to different subproblems to order the actions of these solutions similar to the notion of sidepaths of path reduction. This ordering then guides the search for a solution to the overall problem.

The decomposition of planning problems by state invariants is also present in the object oriented domain specification formalism introduced by McCluskey and Porteous [48]. According to this, states are not defined as collections of facts but as collections of objects, with each object having its own internal status. State invariants are implicitly defined from the requirement that all object attributes are single valued. McCluskey [47] uses *object transition sequences*, i. e., sequences of changes of an object state, to decompose plans. These transition sequences are hereby akin to paths in a c-invariant.

7.1.4 Implementation

Theorem proving has a long tradition in planning. STRIPS [16] by Fikes and Nielsson, the first major planning system, uses the theorem prover QA3 [30] by Green as subsystem to establish the truth of action preconditions. Since then, the use of theorem proving in STRIPS style planning declined in favor of specialized planning algorithms (cf. Section 2.2 on Page 9). Planners for other planning formalisms, e. g., deductive planning [7,8,36] and situation calculus [55] keep close ties with theorem proving. Fronhöfer [21,22] gives a theoretical and experimental comparison between these approaches and STRIPS planning.

Constraint satisfaction is an evident way of representing and solving planning problems. Not surprisingly, many planning systems use this approach, e. g., Graphplan [9], SatPlan [37], and the planner BlackBox [38] of Kautz and Selman. The use of constraint techniques for lifting dates back at least to the planner SNLP [46] of McAllester and Rosenblitt. They give a detailed description of how to lift a planning algorithm to the general level.

7.2 Contributions

In this thesis, we have developed the domain analysis technique path reduction that has the potential to improve the performance of planning systems and the quality of their results. To this end, we have studied a class of state invariants, we call them c-invariants, which can be regarded as transition systems. In analogy to paths through such a transition system, we defined paths through a c-invariant and developed a notion of replaceability of paths in plans. Paths and their mutual dependencies were the starting point for path reduction, which is a technique to construct a path set for a planning problem whose

actions suffice to form a shortest solution to that problem. The main contributions can be summarized as follows.

c-Invariants
We have analyzed, in greater detail as before, the properties of planning domains that result in the existence of c-invariants. In particular, we examined overlapping c-invariants, their destruction, the interaction between different c-invariants, and the interaction between a c-invariant and the part of a planning domain that is not structured by c-invariants. Furthermore, we defined paths and sidepaths through c-invariants which we can regard as relaxation of solutions to the overall planning problem.

Path Replacement
Paths through a given c-invariant are similar across different solutions to a planning problem. We have employed this similarity of paths to develop a general notion of path replaceability. Path replacement hereby balances three desired properties that are mutually exclusive to some extent, namely large number of replaceable paths, wide applicability, and simplicity of application.

Path replacement extends the common notions of replaceable action sequences in two respects: It abstracts from some properties of the replaced sequence and it allows to replace non-continuous partial sequences of plans that can have a conflict. As a result, path replaceability identifies action sequences as replaceable that were disregarded before.

Path Reduction
Another contribution of this thesis is the development of the domain analysis technique path reduction that reduces planning problems by eliminating irrelevant actions. It operates on c-invariants that divide a planning problem into subproblems such that a plan can be considered as the combination of paths, i. e., solutions for these subproblems. Starting from paths and their replacement, path reduction finds sets of paths, whose actions are sufficient to find a shortest solution to the problem. Other techniques to detect irrelevant actions are often incomplete or do not preserve an optimal solution in all cases.

Implementation
Our implementation of path reduction uses a non-ground representation by applying deductive techniques, i. e., theorem proving. It is based on a constraint solver with a theorem prover as subsystem, thus minimizing the use of the latter. Constraint techniques and the use of theorem proving have a long tradition in planning. Nevertheless, the standard techniques are not sufficient to implement path reduction, so that we extended them in a novel way. We know of no system that uses this approach.

7.3 Conclusions

In this work, we have seen that path reduction has the potential of reducing planning problems in novel ways, as path replacement allows to identify irrelevant actions on the basis of non-continuous partial sequences of solutions. Furthermore, path reduction is complete and optimal, and does not require grounding.

On the other hand, path replacement can only be applied to planning problems for which c-invariants have been identified. Furthermore, the reduction power of path reduction does not suffice to yield useful results on many planning domains, even if their c-invariants are known. These results are a consequence of our aim of a complete and optimal technique, as such properties imply either high computational complexity or a weak reduction.

In conclusion, we can say that our development of path replacement yielded interesting results, especially as a non-ground technique, but it is doubtful whether path reduction in its current form can be competitive. Nevertheless, after realizing the noted improvements and trading in completeness and optimality for performance and reduction strength, the approach of reducing planning problems by the exploitation of paths is still promising.

7.4 Further Work

Our work allows a number of extensions and poses interesting open research questions.

Path replacement identifies many replaceable paths, i. e., action sequences. Nevertheless, it can be further generalized: Firstly, it is possible to use a notion of destruction that is based on segments, which would allow to replace a path with destroying actions by a path without. Secondly, we can improve path replacement by covering the case of cascading joinings and disappearances of paths. Both extensions increase the number of possible replacements.

The acceptance of backwards paths is one reason why the current version of path reduction has little effect on planning domains that are currently used to evaluate planning systems. The identification and rejection of backwards paths would at the same time reveal the order in which a solution passes through the facts of c-invariants. Therefore, it is interesting and valuable to extend type-5 path structures so that backwards paths are no longer considered.

The current version of path reduction is aimed at the completeness and the preservation of optimality. Other reduction techniques, e. g., RIFO and RedOp, do not preserve these properties. How competitive is a version of path reduction which trades in these properties for performance, i. e., for runtime, for reduction strength, or for both?

This work shows how to lift a ground preprocessing technique that preserves completeness to the general level. RedOp is another technique with these properties. How can we use the same approach to lift RedOp to the general level? Is the generalized version competitive?

The presented version of path reduction computes a reduced planning problem that is accepted by any planner that accepts the input problem. As the actual output of path reduction is a set of paths, it is an interesting question how to build a planning system that accepts a set of paths instead a set of actions.

Appendix A

The Loop Domain

The loop domain consists of a two-dimensional grid, where each node of the grid is represented by a ground instance of fact *node*(*?x*, *?y*) (cf. Figure 6.6 on Page 114). The coordinates of each node are hereby pairs of objects from a type t. The grid is subdivided into four segments, termed segments 1 to 4, by partitioning t into subtypes t_1 and t_2 of equal size. These segments are comprised by nodes with coordinates from $t_1 \times t_1$, $t_1 \times t_2$, $t_2 \times t_2$, and $t_2 \times t_1$, respectively. Objects in t_1 and t_2 are denoted by a and b, respectively. For example, a_0 is an object in type t_1, b_1 is in type t_2, and $node(a_0, b_1)$ is in segment 2. Similarly, $node(a_0, a_0)$, $node(b_1, b_1)$, and $node(b_1, a_0)$ are in segments 1, 3, and 4, respectively.

The set of all nodes is a c-invariant and the *active node* is the active fact of this c-invariant in a state. Actions allow to *move* the active node along one of the two dimensions. In other words, each action allows to the change one of the two coordinates of the active node, so that a move of the active node to a node that differs in both dimensions requires at least two actions. Changing segments is allowed counterclockwise only, so that revisiting a segment requires to pass through the other three segments. Within a segment, the move of the active node is unrestricted.

A.1 The Unreduced Loop Domain

The unreduced loop domain comprises ten actions and two c-invariants. The first four actions move the active node within a segment. Hereby, the naming of the actions indicates the moves that it enables; for example *move-22-t* allows to move the active node within either segment 3 or segment 4, i. e., either between two nodes with coordinates in $t_2 \times t_2$ or between two nodes with coordinates in $t_2 \times t_1$. Hereby "*22*" stands for a move from t_2 to t_2 in the first coordinate and "*t*" stands for an arbitrary but fixed second coordinate. Actions five to eight allow to change the segment of the active node in the order $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 1$. For example *move-21-1* moves the active node from segment 4 to segment 1. The sole purpose of actions *o-q1* and *o-q2* is to require a solution to a grid problem to pass through $node(b_0, a_1)$ and $node(b_2, a_3)$.

```
(define (domain loop)
(:requirements :strips :typing)
(:types t1 t2 - t)
```

```
(:constants a1 a3 b0 b2)
(:predicates (node ?x ?y - t) (q1 ?x) (q2 ?x) (q3 ?x))

(:action move-11-t
    :parameters  (?x1 - t1 ?x2 - t1 ?y - t)
    :precondition (and (node ?x1 ?y))
    :effect (and (node ?x2 ?y) (not (node ?x1 ?y))))

(:action move-22-t
    :parameters  (?x1 - t2 ?x2 - t2 ?y - t)
    :precondition (and (node ?x1 ?y))
    :effect (and (node ?x2 ?y) (not (node ?x1 ?y))))

(:action move-t-11
    :parameters  (?x - t ?y1 - t1 ?y2 - t1)
    :precondition (and (node ?x ?y1))
    :effect (and (node ?x ?y2) (not (node ?x ?y1))))

(:action move-t-22
    :parameters  (?x - t ?y1 - t2 ?y2 - t2)
    :precondition (and (node ?x ?y1))
    :effect (and (node ?x ?y2) (not (node ?x ?y1))))

(:action move-1-12
    :parameters  (?x - t1 ?y1 - t1 ?y2 - t2)
    :precondition (and (node ?x ?y1))
    :effect (and (node ?x ?y2) (not (node ?x ?y1))))

(:action move-12-2
    :parameters  (?x1 - t1 ?x2 - t2 ?y - t2)
    :precondition (and (node ?x1 ?y))
    :effect (and (node ?x2 ?y) (not (node ?x1 ?y))))

(:action move-2-21
    :parameters  (?x - t2 ?y1 - t2 ?y2 - t1)
    :precondition (and (node ?x ?y1))
    :effect (and (node ?x ?y2) (not (node ?x ?y1))))

(:action move-21-1
    :parameters  (?x1 - t2 ?x2 - t1 ?y - t1)
    :precondition (and (node ?x1 ?y))
    :effect (and (node ?x2 ?y) (not (node ?x1 ?y))))

(:action o-q1
    :parameters  (?x - t)
    :precondition (and (q1 ?x) (node b0 a1))
    :effect (and (q2 ?x) (not (q1 ?x))))
```

```
(:action o-q2
    :parameters  (?x - t)
    :precondition (and (q2 ?x) (node b2 a3))
    :effect (and (q3 ?x) (not (q2 ?x))))

(:invariant :tag cinvariant
  :set-constraint (exactly 1
     (setof :vars (?x ?y - t) (node ?x ?y))))

(:invariant
  :tag cinvariant
  :vars (?x - t)
  :set-constraint (exactly 1
     (setof (q1 ?x))
     (setof (q2 ?x))
     (setof (q3 ?x))))
)
```

A.2 Two Problems

We use two loop problems for experiments. They differ only in their size: The small problem has the grid dimension 4×4 and the large one 15×15. The additional nodes of the larger instance are unused.

```
(define (problem loop_small)
(:domain loop)

(:objects a0 a1 a2 a3 - t1
          b0 b1 b2 b3 - t2)
(:init (node a0 a0) (q1 a0))
(:goal (and (node b3 b3) (q3 a0)))
)

(define (problem loop_large)
(:domain loop)

(:objects a0 a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15 - t1
          b0 b1 b2 b3 b4 b5 b6 b7 b8 b9 b10 b11 b12 b13 b14 b15 - t2)
(:init (node a0 a0) (q1 a0))
(:goal (and (node b3 b3) (q3 a0)))
)
```

A.3 The Reduced Loop Domain

The application of path reduction partially instantiates the actions of the loop domain. As a result, the number of move actions increases from 8 in the original domain to 20 in

the reduced one. At the same time, two of the original actions are no longer present, e. g., `move-11-t` and the number of ground moves for the large problem decreases from 42300 to 200.

```
(define (domain loop)
(:requirements :strips :typing)
(:types t1 t2 - t)
(:constants a0 a1 a3 b0 b2 b3)
(:predicates (node ?x ?y - t)
             (q1 ?x) (q2 ?x) (q3 ?x))

(:action move-22-t-1
    :parameters  ()
    :precondition (node b2 a3)
    :effect (and (node b3 a3) (not (node b2 a3))))

(:action move-22-t-2
    :parameters  ()
    :precondition (node b2 a3)
    :effect (and (node b0 a3) (not (node b2 a3))))

(:action move-22-t-3
    :parameters  ()
    :precondition (node b0 a1)
    :effect (and (node b3 a1) (not (node b0 a1))))

(:action move-22-t-4
    :parameters  ()
    :precondition (node b0 a1)
    :effect (and (node b2 a1) (not (node b0 a1))))

(:action move-t-11-1
    :parameters  ()
    :precondition (node b2 a0)
    :effect (and (node b2 a3) (not (node b2 a0))))

(:action move-t-11-2
    :parameters  ()
    :precondition (node b0 a0)
    :effect (and (node b0 a1) (not (node b0 a0))))

(:action move-t-11-3
    :parameters  ()
    :precondition (node b0 a3)
    :effect (and (node b0 a1) (not (node b0 a3))))
```

```
(:action move-t-11-4
    :parameters  ()
    :precondition (node b2 a1)
    :effect (and (node b2 a3) (not (node b2 a1))))

(:action move-1-12-1
    :parameters  (?y2 - t2)
    :precondition (node a0 a0)
    :effect (and (node a0 ?y2) (not (node a0 a0))))

(:action move-1-12-2
    :parameters  (?x - t1)
    :precondition (node ?x a0)
    :effect (and (node ?x b3) (not (node ?x a0))))

(:action move-1-12-3
    :parameters  (?x - t1)
    :precondition (node ?x a3)
    :effect (and (node ?x b3) (not (node ?x a3))))

(:action move-1-12-4
    :parameters  (?x - t1)
    :precondition (node ?x a1)
    :effect (and (node ?x b3) (not (node ?x a1))))

(:action move-12-2-1
    :parameters  (?y - t2)
    :precondition (node a0 ?y)
    :effect (and (node b3 ?y) (not (node a0 ?y))))

(:action move-12-2-2
    :parameters  (?y - t2)
    :precondition (node a0 ?y)
    :effect (and (node b2 ?y) (not (node a0 ?y))))

(:action move-12-2-3
    :parameters  (?y - t2)
    :precondition (node a0 ?y)
    :effect (and (node b0 ?y) (not (node a0 ?y))))

(:action move-12-2-4
    :parameters  (?x1 - t1)
    :precondition (node ?x1 b3)
    :effect (and (node b3 b3) (not (node ?x1 b3))))
```

```
(:action move-2-21-1
    :parameters  (?y1 - t2)
    :precondition (node b2 ?y1)
    :effect (and (node b2 a3) (not (node b2 ?y1))))

(:action move-2-21-2
    :parameters  (?y1 - t2)
    :precondition (node b0 ?y1)
    :effect (and (node b0 a1) (not (node b0 ?y1))))

(:action move-21-1-1
    :parameters  (?x2 - t1)
    :precondition (node b2 a3)
    :effect (and (node ?x2 a3) (not (node b2 a3))))

(:action move-21-1-2
    :parameters  (?x2 - t1)
    :precondition (node b0 a1)
    :effect (and (node ?x2 a1) (not (node b0 a1))))

(:action o-q1
    :parameters  (?x - t)
    :precondition (and (q1 ?x) (node b0 a1))
    :effect (and (q2 ?x) (not (q1 ?x))))

(:action o-q2
    :parameters  (?x - t)
    :precondition (and (q2 ?x) (node b2 a3))
    :effect (and (q3 ?x) (not (q2 ?x))))

(:invariant :tag cinvariant
  :set-constraint (exactly 1
     (setof :vars (?x ?y - t) (node ?x ?y))))

(:invariant
  :tag cinvariant
  :vars (?x - t)
  :set-constraint (exactly 1
     (setof (q1 ?x))
     (setof (q2 ?x))
     (setof (q3 ?x)))))
```

Index

Bibliography

[1] James Allen, James Hendler, and Austin Tate, editors. *Readings in Planning*. Morgan Kaufmann, 1990.

[2] José Luis Ambite and Craig A. Knoblock. Planning by rewriting: Efficiently generating high-quality plans. In B. Kuipers and B. Webber, editors, *Proc. 14th National Conference on Artificial Intelligence*, pages 706–713, Providence, USA, July 1997. AAAI Press/MIT Press.

[3] José Luis Ambite and Craig A. Knoblock. Planning by rewriting. *Journal of Artificial Intelligence Research*, 15:207–261, September 2001.

[4] José Luis Ambite, Craig A. Knoblock, and Steven Minton. Learning plan rewriting rules. In S. Chien, S. Kambhampati, and C. A. Knoblock, editors, *Proc. 5th Conference on Artificial Intelligence Planning & Scheduling*, pages 3–12, Breckenridge, USA, April 2000. AAAI Press.

[5] Fahiem Bacchus. Subset of PDDL for the AIPS 2000 planning competition. `http://www.cs.toronto.edu/aips2000/pddl-subset.ps`, 2000.

[6] Fahiem Bacchus and Yee Whye Teh. Making forward chaining relevant. In R. Simmons, M. Veloso, and S. Smith, editors, *Proc. 4th Conference on Artificial Intelligence Planning Systems*, pages 54–61, Pittsburgh, USA, June 1998. AAAI PRESS/MIT Press.

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

[8] Wolfgang Bibel. Let's plan it deductively! *Artificial Intelligence*, 103(1-2):183–208, 1998.

[9] Avrim Blum and Merrick Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90:281–300, 1997.

[10] Ronen Brafman. On reachability, relevance, and resolution in the planning as satisfiability approach. *Journal of Artificial Intelligence Research*, 14:1–28, 2001.

[11] Tom Bylander. The computational complexity of STRIPS planning. *Artificial Intelligence*, 69:165–204, 1994.

[12] Clive Dawson and Laurent Siklóssy. The role of preprocessing in problem solving systems. In R. Reddy, editor, *Proc. 5th International Joint Conference on Artificial Intelligence*, pages 465–471, Cambridge, USA, August 1977. William Kaufmann.

[13] Stefan Edelkamp. Planning with pattern databases. In A. Cesta and D. Borrajo, editors, *Proc. 6th European Conference on Planning*, Lecture Notes in Artificial Intelligence, New York, September 2001. Springer.

[14] Stefan Edelkamp and Malte Helmert. Exhibiting knowledge in planning problems to minimize state encoding length. In M. Fox and S. Biundo, editors, *Proc. 5th European Conference on Planning*, volume 1809 of *Lecture Notes in Artificial Intelligence*, pages 135–147, New York, 1999. Springer.

[15] Stefan Edelkamp and Malte Helmert. The model checking integrated planning system. *AI magazine*, 22(3):67–71, 2001.

[16] Richard E. Fikes and Nils J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[17] Norman Y. Foo, Abhaya Nayak, Maurice Pagnucco, Pavleo Peppas, and Yan Zhang. Action localness, genericity and invariants in STRIPS. In M. Pollack, editor, *Proc. 14th International Joint Conference on Artificial Intelligence*, pages 549–554, Nagoya, Japan, August 1997. Morgan Kaufmann.

[18] Maria Fox and Derek Long. The automatic inference of state invariants in TIM. *Journal of Artificial Intelligence Research*, 9:367–421, December 1998.

[19] Maria Fox and Derek Long. Utilizing automatically inferred invariants in graph construction and search. In S. Chien, S. Kambhampati, and C. A. Knoblock, editors, *Proc. 5th Conference on Artificial Intelligence Planning & Scheduling*, pages 102 – 111, Breckenridge, USA, April 2000. AAAI Press.

[20] Maria Fox and Derek Long. Hybrid STAN: Identifying and managing combinatorial optimisation sub-problems in planning. In B. Nebel, editor, *Proc. 16th International Joint Conference on Artificial Intelligence*, pages 445–452, Seattle, USA, August 2001. Morgan Kaufmann.

[21] Bertram Fronhöfer. Situational calculus, linear connection proofs and STRIPS-like planning: An experimental comparison. In P. Miglioli, U. Moscato, D. Mundici, and M. Ornaghi, editors, *Proceedings of 5th International Workshop on Theorem Proving with Analytic Tableaux and Related Methods*, volume 1071 of *Lecture Notes in Computer Science*, pages 193–209, Terrasini, Palermo, Italy, May 1996. Springer.

[22] Bertram Fronhöfer. Plan generation with the linear connection method. *INFORMATICA, Lithuanian academy of sciences*, 8(1):3–22, 1997.

[23] Michael R. Garey and David. S. Johnson. *Computers and Intractability: A Guide to the Theory of $\mathcal{NP}$-Completeness.* Freeman, San Francisco, CA, USA, 1979.

[24] Alfonso Gerevini and Lenhard Schubert. Inferring state constraints for domain-independent planning. In *Proc. 15th National Conference on Artificial Intelligence*, pages 905–912, Madison, USA, July 1998. AAAI Press/MIT Press.

[25] Alfonso Gerevini and Lenhard Schubert. Discovering state constraints in DISCOPLAN: Some new results. In H. Kautz and B. Porter, editors, *Proc. 17th National Conference on Artificial Intelligence*, pages 761–767, Austin, USA, July 2000. AAAI Press/MIT Press.

[26] Alfonso Gerevini and Lenhard Schubert. Extending the types of state constraints discovered by DISCOPLAN. In M. Fox, editor, *Proceedings of the Workshop at AIPS on Analyzing and Exploiting Domain Knowledge for Efficient Planning*, pages 4–11, Breckenridge, USA, April 2000.

[27] Alfonso Gerevini and Ivan Serina. Fast planning through greedy action graphs. In *Proc. 16th National Conference on Artificial Intelligence*, pages 503–510, Orlando, USA, July 1999. AAAI Press/MIT Press.

[28] Alfonso Gerevini and Ivan Serina. LPG: a planner based on local search for planning graphs with action costs. In M. Ghallab, J. Hertzberg, and P. Traverso, editors, *Proc. 6th Conference on Artificial Intelligence Planning & Scheduling*, pages 281–290. AAAI press, April 2002.

[29] Patrice Godefroid and Froduald Kabanza. An efficient reactive planner for synthesizing reactive plans. In *Proc. 9th National Conference on Artificial Intelligence*, pages 640–645, Anaheim, USA, July 1991. AAAI Press/MIT Press.

[30] Cordell Green. Theorem proving by resolution as a basis for question-answering systems. *Machine Intelligence*, 4:183–208, 1969.

[31] Reiner Hähnle, Manfred Kerber, and Chrisoph Weidenbach. Common syntax of the DFG-Schwerpunktprogramm "Deduktion". Internal Report 10/96, Universität Karlsruhe, Fakultät für Informatik, 1996.

[32] Patrik Haslum and Peter Jonsson. Planning with reduced operator sets. In S. Chien, S. Kambhampati, and C. A. Knoblock, editors, *Proc. 5th Conference on Artificial Intelligence Planning & Scheduling*, pages 150–158, Breckenridge, USA, April 2000. AAAI Press.

[33] Patrik Haslum and Ulrich Scholz. Domain knowledge in planning: Representation and use. In D. Long, D. McDermott, and S. Thiébaux, editors, *ICAPS'03 Workshop on PDDL*, pages 69–78, May 2003.

[34] Jörg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

[35] Jörg Hoffmann and Bernhard Nebel. RIFO revisited: Detecting relaxed irrelevance. In A. Cesta and D. Borrajo, editors, *Proc. 6th European Conference on Planning*, Lecture Notes in Artificial Intelligence, pages 325–336, New York, September 2001. Springer.

[36] Steffen Hölldobler and Josef Schneeberger. A new deductive approach to planning. *New Generation Computing*, 8:225–244, 1990.

[37] Henry Kautz and Bart Selman. Planning as satisfiability. In B. Neumann, editor, *Proc. 10th European Conference on Artificial Intelligence*, pages 359–363, Vienna, Austria, August 1992. John Wiley & Sons, Ltd.

[38] Henry Kautz and Bart Selman. Blackbox: A new approach to the application of theorem proving to problem solving. In *AIPS-98 Workshop on Planning as Combinatorial Search*, pages 58–60, Pittsburgh, USA, June 1998.

[39] Jana Koehler. RIFO within IPP. Technical Report 126, Institute for Computer Science, University Freiburg, Germany, 1999.

[40] Jana Koehler and Jörg Hoffmann. Handling of inertia in a planning system. Technical Report 122, Institute for Computer Science, University Freiburg, Germany, 1999.

[41] Jana Koehler and Jörg Hoffmann. On the instantiation of ADL operators involving arbitrary first-order formulas. In *Proceedings of ECAI-00 Workshop on New Results in Planning, Scheduling, and Design*, Berlin, Germany, August 2000.

[42] Jana Koehler, Bernhard Nebel, Jörg Hoffmann, and Yannis Dimopoulos. Extending planning graphs to an ADL subset. In S. Steel and R. Alami, editors, *Proc. 4th European Conference on Planning*, volume 1348 of *Lecture Notes in Computer Science*, pages 273–285, Toulouse, France, September 1997. Springer.

[43] Derek Long and Maria Fox. Efficient implementation of the plan graph in STAN. *Journal of Artificial Intelligence Research*, 10:87–115, February 1999.

[44] Derek Long and Maria Fox. Extracting route-planning: First steps in automatic problem decomposition. In M. Fox, editor, *Proceedings of AIPS Workshop on Analysing and Exploiting Domain Knowledge for Efficient Planning*, pages 22–28, Breckenridge, USA, 2000.

[45] Derek Long and Maria Fox. Recognizing and exploiting generic types in planning domains. In S. Chien, S. Kambhampati, and C. A. Knoblock, editors, *Proc. 5th Conference on Artificial Intelligence Planning & Scheduling*, pages 196 – 205, Breckenridge, USA, April 2000. AAAI Press.

[46] David McAllester and David Rosenblitt. Systematic nonlinear planning. In *Proc. 9th National Conference on Artificial Intelligence*, volume 2, pages 634–639, Anaheim, USA, July 1991. AAAI Press/MIT Press.

[47] Thomas Leo McCluskey. Object transition sequences: A new form of abstraction for HTN planners. In S. Chien, S. Kambhampati, and C. A. Knoblock, editors, *Proc. 5th Conference on Artificial Intelligence Planning & Scheduling*, pages 216–225, Breckenridge, USA, April 2000. AAAI Press.

[48] Thomas Leo McCluskey and Julie Porteous. Engineering and compiling planning domain models to promote validity and efficiency. *Artificial Intelligence*, 95:1–65, 1997.

[49] Drew McDermott, Malik Ghallab, Adele Howe, Craig A. Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wikins. PDDL - the planning domain definition language. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, 1998.

[50] Bernhard Nebel, Yannis Dimopoulos, and Jana Koehler. Ignoring irrelevant facts and operators in plan generation. In S. Steel and R. Alami, editors, *Proc. 4th European Conference on Planning*, volume 1348 of *Lecture Notes in Computer Science*, pages 338–350, Toulouse, France, September 1997. Springer.

[51] XuanLong Nguyen and Subbarao Kambhampati. Reviving partial order planning. In B. Nebel, editor, *Proc. 16th International Joint Conference on Artificial Intelligence*, pages 459–466, Seattle, Washington, August 2001. Morgan Kaufmann.

[52] Ioannis Refanidis and Ioannis Vlahavas. SSPOP: A state-space partial-order planner. In *World Multiconference on Systemics, Cypernetics and Informatics*, pages 240–247, Orlando, USA, July 1999.

[53] Ioannis Refanidis and Ioannis Vlahavas. Exploiting state constraints in heuristic state-space planning. In S. Chien, S. Kambhampati, and C. A. Knoblock, editors, *Proc. 5th Conference on Artificial Intelligence Planning & Scheduling*, pages 363–370, Breckenridge, USA, April 2000. AAAI Press.

[54] Ioannis Refanidis and Ioannis Vlahavas. The GRT planning system: Backward heuristic construction in forward state-space planning. *Journal of Artificial Intelligence Research*, 15:115–161, August 2001.

[55] Raymond Reiter. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press, Cambridge, Massachusetts, 2001.

[56] Jussi Rintanen. An iterative algorithm for synthesizing invariants. In H. Kautz and B. Porter, editors, *Proc. 17th National Conference on Artificial Intelligence*, pages 806–811, Austin, USA, July 2000. AAAI Press/MIT Press.

[57] John A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, 1965.

[58] Stuart Russell and Peter Norwig. *Artificial Intelligence, A Modern Approach*. Prentice Hall, 1995.

[59] Earl D. Sacerdoti. The nonlinear nature of plans. In *Proc. 4th International Joint Conference on Artificial Intelligence*, pages 206 – 214, Tbilisi, USSR, September 1975.

[60] Ulrich Scholz. Planning by local search. Master's thesis, Darmstadt University of Technology, Darmstadt, Germany, December 1997.

[61] Ulrich Scholz. Action constraints for planning. In S. Biundo and M. Fox, editors, *Proc. 5th European Conference on Planning*, volume 1809 of *Lecture Notes in Artificial Intelligence*, pages 148–160, New York, September 1999. Springer.

[62] Stephan Schulz. E – a brainiac theorem prover. *Journal of AI Communications*, 15(2/3):111–126, 2002.

[63] Stephen Soderland and Daniel S. Weld. Evaluating non-linear planning. Technical Report TR-91-02-03, University of Washington, Department of Computer Science and Engineering, Seattle, Washington, 1991.

[64] Austin Tate. Interacting goals and their use. In *Proc. 4th International Joint Conference on Artificial Intelligence*, pages 215–218, Tbilisi, USSR, September 1975.

[65] Larry Taylor and Richard Korf. Pruning duplicate nodes in depth-first search. In *Proc. 11th National Conference on Artificial Intelligence*, pages 756–761. AAAI Press/MIT Preess, July 1993.

[66] Dimitris Vrakas, Grigorios Tsoumakas, Nick Bassiliades, and Ioannis Vlahavas. Learning rules for adaptive planning. In E. Giunchiglia, N. Muscettola, and D. Nau, editors, *Proc. 13th International Conference on Automated Planning and Scheduling*, pages 82–91, Trento, Italy, June 2003.

[67] Richard Waldinger. Achieving several goals simultaneously. In E.W. Elcock and D. Michie, editors, *Machine Intelligence 8*, pages 94–138. Ellis Horwood, Chichester, England, 1977. (also in: [1]).

[68] Mark Wallace, Stefano Novello, and Joachim Schimpf. *ECL^iPS^e: A platform for constraint logic programming*. IC-Parc, William Penney Laboratory, Imperial College, London SW7 2AZ, August 1997.

[69] David Warren. WARPLAN: A system for generating plans. Memo 76, Department of Computational Logic, University of Edinburgh, Edinburgh, Scotland, 1974. (also in: [1]).

[70] Håkan Younes and Reid Simmons. On the role of ground actions in refinement planning. In M. Ghallab, J. Hertzberg, and P. Traverso, editors, *Proc. 6th Conference on Artificial Intelligence Planning & Scheduling*, pages 54–61. AAAI press, April 2002.

[71] Yan Zhang and Norman Y. Foo. Deriving invariants and constraints from action theories. *Fundamenta Informatica*, 30:23–41, 1996.