Theory of Hybrid Systems Informatik 2

RWTH AACHEN UNIVERSITY

The present work was submitted to the LuFG Theory of Hybrid Systems

BACHELOR OF SCIENCE THESIS

A CEGAR APPROACH FOR EFFICIENT PLANNING

László Dirks

Examiners:
Prof. Dr. Erika Ábrahám
Prof. Gerhard Lakemeyer, Ph.D.

Additional Advisor:
Francesco Leofante, Ph.D

Aachen, September 22, 2021

Abstract

We present a counterexample guided abstraction refinement (CEGAR) approach for numeric planning. We modify existing SAT-based approaches by omitting certain parts of the encoding. This allows for a more compact plan representation. To derive valid plans from the abstraction we propose multiple methods of refinement. In addition, we modify an existing search algorithm by applying an incremental solving strategy to make use of the encoding's structure. We provide an implementation extending OMTPlan and evaluate it on standard benchmarks for numeric planning problems taken from the literature. The empirical evaluation shows that the incremental solving strategy is an improvement to the original method on many domains. Our CEGAR approach outperforms state-of-the-art methods on smaller problem instances. Scalability remains challenging on larger instances.

Erklärung

Hiermit versichere ich, dass ich die vorgelegte Arbeit selbstständig verfasst und noch nicht anderweitig zu Prüfungszwecken vorgelegt habe. Alle benutzten Quellen und Hilfsmittel sind angegeben, wörtliche und sinngemäße Zitate wurden als solche gekennzeichnet.

Aachen, den 22. September 2021

Acknowledgements

First and foremost, I wish to express my sincere thanks to Prof. Dr. Erika Ábrahám and Francesco Leofante, Ph.D. for their dedicated involvement and helpful expertise. I am also grateful to Prof. Gerhard Lakemeyer, Ph.D. for being my second supervisor.

Contents

1 Introduction **9**
1.1 AI Planning . . . 9
1.2 Counterexample Guided Abstraction Refinement . . . 9
1.3 Outline of the Thesis . . . 10

2 Preliminaries **11**
2.1 Numeric Planning . . . 11
2.2 Encoding Parallel Plans . . . 12

3 A CEGAR Approach for Efficient Planning **15**
3.1 Incremental Parallel Search . . . 15
3.2 A Relaxation . . . 16
3.3 Methods for Testing Sequentializability . . . 21

4 Experimental Results **31**
4.1 Comparisons . . . 31
4.2 Analysis . . . 31

5 Conclusion **39**
5.1 Summary . . . 39
5.2 Future Work . . . 39

Bibliography **41**

Chapter 1

Introduction

1.1 AI Planning

Planning is a branch of the artificial intelligence research field exploring the problem of enabling an agent to find a plan consisting of a sequence of actions. Given an initial state the entity should reach a state that satisfies a goal condition by executing those actions. In this context the world in a planning problem instance is modelled by a set of state variables. The assignment of values to all state variables defines a state of the world.

In order to establish a standard in the planning community for an easier exchange of planning domains and comparability of research results a *Planning Domain Description Language* (`PDDL`) was introduced in 1998 by Drew McDermott. Since the definition of the original `PDDL`, which only allowed propositional variables, several updates to the language up to version 3.1 were made. Further, several modelling languages such as NDL [Rin15] have been developed. The planning problems we consider in this work can be encoded in `PDDL2.1` [FL11], which enables the definition of domains for reasoning about numeric quantities.

The earliest methods for solving planning problems are based on explicit state-space search. Algorithms using this approach use the strategy to explicitly generate states which can be reached from the initial state. Such algorithms for example are uniform search algorithms like depth first search or heuristic based algorithms such as A* and variants of it [SS11].

A different approach for planning is the reduction of subproblems to SAT or SMT. The underlying idea of this approach is to encode the existence of a plan of some finite length for some planning problem into a formula. The encoded formula is then solved by an off-the-shelf SAT or SMT solver [SD05]. State of the art planning tools, which are based on further developments of this approach, include Springroll [SRHT16] and Rantanplan [BEV16].

1.2 Counterexample Guided Abstraction Refinement

OMTPlan [LGÁT20] is a planning tool, primarily developed to find cost optimal plans for numeric planning problems where actions may have state dependent or constant costs. The implementation also provides a search algorithm for a satisficing (not

optimal) search, which relies on a state based encoding of the existence of a plan of some length. This encoding allows the parallel execution of multiple actions and therefore contains encodings of mutexes which ensure that these parallel plans can be converted to valid sequential plans. Allowing this parallelism has the advantage that longer plans can be found at an earlier stage in the search. The syntactically derived mutexes used in the encoding however have the drawback that they restrict some of the possible parallelism and cause an asymptotically quadratically long encoding in the worst case.

Our new approach therefore is to encode an abstraction of the existence of a plan by omitting these mutexes. Subsequently the abstraction will be refined during the search process, in case a model of the abstraction is found which can not be converted to a sequential plan. The motivation is that doing so, we have an initially asymptotically linear encoding and need fewer iterations in the search process as more parallelism is allowed.

1.3 Outline of the Thesis

We introduce a formal definition of the planning problems we intend to solve in Chapter 2, Section 2.1 and make a detailed presentation of the SMT-encoding used in the algorithm on which our approach is based in Section 2.2.

In Chapter 3, we propose a technical improvement of the original algorithm in Section 3.1 and describe our CEGAR approach in Section 3.2 in detail. The chapter is concluded with a brief proof of soundness and completeness of the algorithm.

With an empirical evaluation in Chapter 4 we compare our new approach to both the original algorithm and another state of the art planning tool. Furthermore we analyse the encodings and runtimes of the subroutines in our algorithm.

In Chapter 5 we conclude our findings through a recapitulation and discuss how the strengths and weaknesses of our approach, observed in the previous chapter, could be subject of future work.

Chapter 2

Preliminaries

2.1 Numeric Planning

In the following a *planning problem* is considered to be from a fragment of numeric planning, which can be expressed in `PDDL2.1` level 2 [FL11]. We follow the notation introduced in [LGÁT20]. A *planning problem* is defined as a tuple

$$\Pi = (\mathcal{V}_{\mathbb{B}}, \mathcal{V}_{\mathbb{Q}}, A, I, G) \tag{2.1}$$

where $\mathcal{V}_{\mathbb{B}} \cap \mathcal{V}_{\mathbb{Q}} = \emptyset$ with $\mathcal{V}_{\mathbb{B}}$ being the set of propositional variables and $\mathcal{V}_{\mathbb{Q}}$ being the set of numeric variables in the problem Π. Let $dom(v)$ denote the domain of a variable v. We set $\forall v \in \mathcal{V}_{\mathbb{B}} : dom(v) = \mathbb{B}$ and $\forall v \in \mathcal{V}_{\mathbb{Q}} : dom(v) = \mathbb{Q}$. A *state* of a planning problem Π is defined through an assignment of all variables $\mathcal{V} := \mathcal{V}_{\mathbb{B}} \cup \mathcal{V}_{\mathbb{Q}}$. Therefore a state can formally be defined as a function $s : \mathcal{V} \rightarrow (\mathbb{B} \cup \mathbb{Q})$ with $s(v) \in \mathbb{B}$ for all $v \in \mathcal{V}_{\mathbb{B}}$ and $s(v) \in \mathbb{Q}$ for all $v \in \mathcal{V}_{\mathbb{Q}}$. The set of all states of Π is denoted as S.

A *propositional* constraint consists of a variable $v \in \mathcal{V}_{\mathbb{B}}$ or the negation $\neg v$ for $v \in \mathcal{V}_{\mathbb{B}}$. Considering the usual linear rational arithmetic structure over the signature $\tau = \{+, -, \cdot, <, \leq, =, \geq, >\}$ with the domain $\mathbb{Q}$, an *arithmetic expression* e is a $\tau - term$ with function symbols from τ, constants in $\mathbb{Q}$ and variables from $\mathcal{V}$. An arithmetic expression is *linear*, if and only if no occurrence of the multiplication operator in e is used to multiply two variables. We only consider linear arithmetic expressions. A *numeric* constraint is a τ-formula $e_1 \sim e_2$ with a relation symbol $\sim \in \{<, \leq, =, \geq, >\}$ and e_1, e_2 being arithmetic expressions. *Constraints* are either numeric or propositional constraints. A *condition* is a finite set of constraints. We use the usual notation $s \models \Phi$ to denote that a mapping s models a set of constraints Φ.

A *propositional assignment* $v := e$ is the assignment of an element $e \in \{\top, \bot\}$, representing true and false, to a propositional variable $v \in \mathcal{V}_{\mathbb{B}}$. We call $v := \top$ an *add effect* and $v := \bot$ a *delete effect*. Accordingly, a *numeric assignment* $v := e$, an assignment to v, consists of a numeric variable $v \in \mathcal{V}_{\mathbb{B}}$ and an arithmetic expression e. The evaluation of a term e under a mapping s is denoted as $[\![e]\!]_s$. We define an *effect* Ψ as a set of assignments containing at most one assignment per variable $v \in \mathcal{V}$.

For some state $s \in S$, given an effect Ψ, the *successor* of s and Ψ is the unique state $s' \in S$ with

$$s'(v) := \begin{cases} [\![e]\!]_s & \text{if } v := e \in \Psi \\ s(v) & \text{else} \end{cases}.$$

Otherwise put, the value of a variable v in the successor state s' corresponds to the application of the assignment $v := e$ in the state s, if there is such an assignment in Ψ and remains unaltered in relation to s otherwise.

In the above definition of a planning problem A is a finite set of actions. Each action $a \in A$ consists of a condition pre_a, called the *precondition*, and an effect eff_a, $a = (pre_a, \mathit{eff}_a)$. In general it is customary to also consider the cost of an action, but here we will remain with this simpler definition as the action's costs have no relevance in the purely satisficing setting of this work.

With e being a linear arithmetic expression and $d \in \mathbb{Q}$ we can distinct between a *constant increment* $v := v + d$, a *constant decrement* $v := v - d$, a *linear increment* $v := v + e$, and a *linear decrement* $v := v - e$. We say that a numeric constraint $e \sim 0$ is *simple* if and only if every assignment in $\cup_{a \in A} \mathit{eff}_a$ of a variable that appears in e is a constant increment or a constant decrement. Analogously, a numeric constraint $e \sim 0$ is *linear* if and only if every assignment in $\cup a \in A \mathit{eff}_a$ of a variable that appears in e is a linear increment or a linear decrement.

An action a is called *applicable* in a state s, if and only if the state satisfies the preconditions of the action, i.e. if and only if for all $\varphi \in pre_a$ it holds that $s \models \varphi$. The unique initial state is defined by a finite set of constraints I, called the *initial condition*. All constraints in I have the form $v = e$, $e \in \mathbb{Q}$ for each $v \in \mathcal{V}_\mathbb{Q}$ and v or $\neg v$ for each $v \in \mathcal{V}_\mathbb{B}$. The set of goal states is defined by a condition G, called the *goal condition*.

With this in mind we can finally come to the definition of the sought-after subject. A plan π_n, with $n \in \mathbb{N}$ denoting the plan's length, is a sequence of actions $\pi_n = \langle a_0, ..., a_{n-1} \rangle$, $a_0, ..., a_{n-1} \in A$ such that there exists a unique sequence of states $s_0, ..., s_n$, where $s_0 \models I$, $s_n \models G$ and for every $i \in \{1, ..., n\}$ it holds that $s_{i-1} \models pre_{a_{i-1}}$ and s_i is the successor of s_{i-1} and $\mathit{eff}_{a_i - 1}$. For the sake of unambiguity we will refer to these plans as *sequential* plans and we call plans of later defined kinds *sequentializable*, if we are able to convert them into a sequential plan.

2.2 Encoding Parallel Plans

Given a planning problem $\Pi = (\mathcal{V}_\mathbb{B}, \mathcal{V}_\mathbb{Q}, A, I, G)$, the search for a plan in [LGÁT20] consists of encoding the existence of a plan of up to a certain length h, which is also being referred to as the *horizon*, and iteratively increasing that horizon to an arbitrary upper bound ub as long as no model is found.

In order to make the encoding more efficient the concept of a *parallel plan* is used. A set of actions $A' \subseteq A$ is *independent*, iff for any variable $v \in V$ where $v := e \in \mathit{eff}_{a_1}$ for some action $a_1 \in A'$ and some term e there is no $a_2 \in A'$, $a_2 \neq a_1$ where $v := e' \in \mathit{eff}_{a_2}$ or v appears in a precondition of a_2. A *parallel plan* of length n

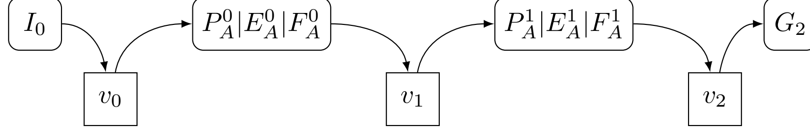

Figure 2.1: Illustration of the state based encoding.

is a sequence of independent sets of actions $\pi_n = \langle A_0, ..., A_{n-1} \rangle$ such that, with $A_i = \{a_{i,j} | 0 < j \leq k_i\}$, the action sequence $\langle a_{0,1}, ..., a_{0,k_0}, ..., a_{n-1,1}, ..., a_{n-1,k_{n-1}} \rangle$ is a sequential plan for Π. Due to the restriction that each set of actions has to be independent, parallel plans in general can be sequentialized by ordering the actions of each action set in any order. The intuition behind enabling this parallelism is that the existence of a parallel plan of horizon h can be encoded in a formula with a similar length as the encoding of the existence of a sequential plan with the same horizon, but the resulting sequentialized plan can have a length of up to $h \cdot |A|$. This potentially allows finding longer plans with less computational expense.

In the following we will describe in detail the encoding used in [LGÁT20]. The existence of a parallel plan with the horizon h for the planning problem Π is translated into a state based encoding. For this purpose $h{+}1$ *state variables* $\{v_i | v \in \mathcal{V}, 0 \leq i \leq h\}$ in the encoding represent $h+1$ different states in a parallel plan. Each item in the sequence that is a plan, is called a ***step***. The state variables correspond to the states before and after each parallel plan step. We use the upper indices t on terms and formulas from Π to annotate them with the plan step to which they belong:

$$\begin{aligned} v^t &: v_t \\ (\neg v)^t &: \neg v^t \\ q^t &: q && \text{for } q \in \mathbb{Q} \\ (e_1 \bullet e_2)^t &: e_1^t \bullet e_2^t && \text{where } \bullet \in \{+, -, \cdot\} \\ (v \sim e)^t &: v^t \sim e^t && \text{where } \sim \in \{<, \leq, =, \geq, >\} \end{aligned}$$

The initial condition I is encoded as:

$$I_0 : \bigwedge_{v=q \in I} v_0 = q \wedge \bigwedge_{v \in I} v_0 \wedge \bigwedge_{\neg v \in I} \neg v_0 \tag{2.2}$$

The goal condition for a certain horizon h is encoded as follows, where $\sim \in \{<, \leq, =, \geq, >\}$:

$$G_h : \bigwedge_{v \sim e \in G} v_h \sim e^h \wedge \bigwedge_{v \in G} v_h \wedge \bigwedge_{\neg v \in G} \neg v_h \tag{2.3}$$

Each step A_t is encoded using propositional variables a^t for each action $a \in A$ that can possibly be executed in one parallel step, with $0 \leq t < h$. Now to encode an action $a \in A$ at the time step t the following subformulas are needed. The execution of the actions implies that all constraints in the precondition of the action are fulfilled:

$$P_a^t : a^t \rightarrow \bigwedge_{\varphi \in pre_a} \varphi^t \tag{2.4}$$

Further, the execution of the action implies the assignments in the action's effect:

$$E_a^t : a^t \rightarrow \bigwedge_{v:=e \in \mathit{eff}_a} v^{t+1} = e^t \wedge \bigwedge_{v:=\top \in \mathit{eff}_a} v^{t+1} \wedge \bigwedge_{v:=\bot \in \mathit{eff}_a} \neg v^{t+1} \tag{2.5}$$

For a more compact notation we define:

$$P_A^t : \bigwedge_{a \in A} P_a^t \tag{2.6}$$

$$E_A^t : \bigwedge_{a \in A} E_a^t \tag{2.7}$$

Be $A_{add}^v \subseteq A$ the set of actions with add effects and $A_{del}^v \subseteq A$ the set of actions with delete effects on a propositional variable $v \in \mathcal{V}_{\mathbb{B}}$. Further be $A_{num}^v \subseteq A$ the set of actions with effects on a numerical variable $v \in \mathcal{V}_{\mathbb{Q}}$. It also needs to be encoded, that the value of a state variable remains unchanged from one step to the next, if no action alters it. This part of the encoding is referred to as the *frame*:

$$F_A^t : \bigwedge_{v \in \mathcal{V}_{\mathbb{Q}}} \big((v^t = v^{t+1}) \vee \bigvee_{a \in A_{num}^v} a^t\big) \tag{2.8}$$

$$\wedge \bigwedge_{v \in \mathcal{V}_{\mathbb{B}}} \Big(\big((\neg v^t \wedge v^{t+1}) \rightarrow \bigvee_{a \in A_{add}^v} a^t\big) \tag{2.9}$$

$$\wedge \big((v^t \wedge \neg v^{t+1}) \rightarrow \bigvee_{a \in A_{del}^v} a^t\big)\Big) \tag{2.10}$$

Now to ensure independence of the set of actions in one parallel step, for each pair of actions $(a_1, a_2) \in A^2$, $a_1 \neq a_2$ it is checked whether

(i) a variable appears both in a precondition of a_1 and in the effect of a_2,

(ii) or a variable appears both in a delete effect of a_1 and in an add effect of a_2,

(iii) or a variable appears both in a numeric assignment in eff_{a_1} and in a numeric assignment in eff_{a_2}.

If any of the above cases hold, the actions are considered to be mutually exclusive. With $A_m \subsetneq A^2$ being the set of all *mutexes*[1], i.e. 2-tuples of mutually exclusive actions, we encode:

$$M^t : \bigwedge_{(a_1, a_2) \in A_m} (\neg a_1^t \vee \neg a_2^t) \tag{2.11}$$

In conclusion, the existence of a parallel plan of length at most h is encoded as:

$$P_\Pi^h : I_0 \wedge G_h \wedge \bigwedge_{0 \leq t < h} \big(F_A^t \wedge P_A^t \wedge E_A^t \wedge M^t\big) \tag{2.12}$$

[1]The set A_m does not include tuples of the form $(a,a), a \in A$, as the encoding implies that an action can only be executed once per step.

Chapter 3

A CEGAR Approach for Efficient Planning

The focus of this work lays on exploring different methods on how the mutex conditions as mentioned in the preliminaries section can be relaxed to allow more parallelism and a simpler encoding. The first section of this chapter however introduces a rather technical alteration of the parallel plan search algorithm, before we explore the relaxation in the second section.

3.1 Incremental Parallel Search

The parallel plan search in [LGÁT20] consists of iteratively constructing the formula P_Π^h for increasing horizons until an upper bound ub is reached, or the underlying solver z3 [BdMNW18] finds a model for the formula, see Algorithm 1.

Algorithm 1 Parallel Search.

Input: Π, ub

```
1:  h := 1
2:  while h ≤ ub do
3:      encoding = P_Π^h
4:      solver = z3.Solver()
5:      solver.add(encoding)
6:      res = solver.check()
7:      if res == sat then
8:          model = solver.model()
9:          return plan(model)
10:     end if
11:     h += 1
12: end while
13: return 'No plan found within bound.'
```

This procedure involves encoding and solving the entire formula for each iteration. However, in the formula for horizon h the encoding of all actions at step $t < h - 1$ and the encoding of the initial state remain unchanged to the encoding at horizon

$h-1$. To speed up the encoding in each step and potentially make use of internally derived clauses in the solver, we updated the parallel search procedure to make use of this *incremental* structure in the encoding. For this purpose we used the incremental solver in z3 [BdMNW18], which allows to set backtracking points (`push`) in-between formula assertions to the solver and remove the assertions after the last backtracking point (`pop`), see Algorithm 2.

Algorithm 2 Incremental Parallel Search.

Input: $\Pi = (\mathcal{V}_{\mathbb{B}}, \mathcal{V}_{\mathbb{Q}}, A, I, G), ub$

1: $h := 1$
2: initial_s := I_0
3: solver = z3.Solver()
4: solver.add(initial_s)
5: solver.push()
6: **while** $h \leq ub$ **do**
7: encoding = $F_A^{h-1} \wedge P_A^{h-1} \wedge E_A^{h-1} \wedge M^{h-1}$
8: solver.add(encoding)
9: solver.push()
10: solver.add(G_h)
11: res = solver.check()
12: **if** res == sat **then**
13: model = solver.model()
14: **return** plan(model)
15: **end if**
16: solver.pop()
17: h += 1
18: **end while**
19: **return** 'No plan found within bound.'

3.2 A Relaxation

The use of mutexes in the encoding to ensure independence as defined in Equation 2.11 restricts the possible parallelism that can be achieved through the encoding of a parallel plan excluding the mutexes. The derived mutexes generally may exclude certain pairs of actions, which do not necessarily interfere in a parallel plan step. Let us illustrate this through a brief example.

Example 3.2.1. Let us consider the following planning problem.

$\Pi = (\mathcal{V}_{\mathbb{B}} = \emptyset, \mathcal{V}_{\mathbb{Q}} = \{x,y\}, A, I = \{x = 0, y = 0\}, G = \{x \geq 1, y \geq 1\})$ with
$A = \{increase_x, increase_y\}$ and
$increase_x = (\{y \leq 1\}, \{x = x + 1\})$
$increase_y = (\{x \leq 1\}, \{y = y + 1\})$

The actions $increase_x$ and $increase_y$ are considered to be mutually exclusive as the numeric variable x appears in the precondition of $increase_y$ and in the effect of $increase_x$. The parallel plan with the smallest horizon using the mutex

encoding from Equation 2.11 therefore has the length 2. One possible solution would be $\pi_{mut} = \langle\{increase_x\},\{increase_y\}\rangle$. If we exclude the mutexes, already the (unique) solution for the horizon 1 would yield a sequentializable solution: $\pi_{rel} = \langle\{increase_x, increase_y\}\rangle$.

We want to abstract from the parallel plan and refine our search, if the solution of the abstraction can not be converted to a valid plan as illustrated in Figure 3.1.

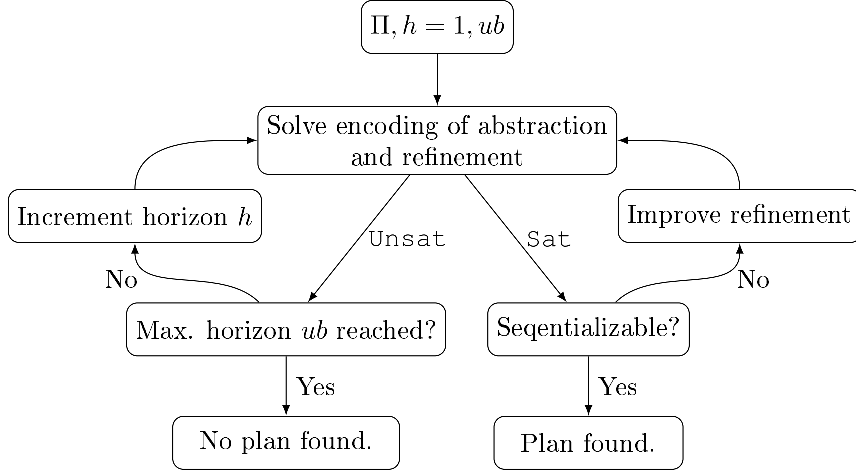

Figure 3.1: Illustration of the CEGAR approach.

To explore possible benefits in a relaxation we define the *relaxed parallel plan.* Let us again assume some planning problem $\Pi = (\mathcal{V}_{\mathbb{B}}, \mathcal{V}_{\mathbb{Q}}, A, I, G)$. We previously defined the successor of a state, given the effect of an action. For the relaxation, we need to transfer the notion of a successor of a state $s \in S$ to the effects of a set of actions A', $\Psi = \bigcup_{a \in A'} \mathit{eff}_a$. We define the successor:

$$s'(v) := \begin{cases} [\![e]\!]_s & \text{if } (v := e) \in \Psi \wedge \neg\exists(v := e') \in \Psi : [\![e]\!]_s \neq [\![e]\!]_s \\ s(v) & \text{if } \neg\exists(v := e) \in \Psi \\ \texttt{undef} & \text{otherwise} \end{cases}$$

Definition 3.2.1 (Relaxed Parallel Plan)**.** *A relaxed parallel plan of length n is a sequence of sets of actions $\pi_n = \langle A_0, ..., A_{n-1}\rangle$, with $A_i = \{a_{i,j} | 0 < j \leq k_i\} \subseteq A$ such that there exists a unique sequence of states $s_0, ..., s_n$, where*

- $s_0 \models I, s_n \models G$
- *for every $i \in \{1, ..., n\}$ and $a_{i-1,j} \in A_{i-1}$ it holds that $s_{i-1} \models pre_{a_{i-1,j}}$*
- *s_i is the well-defined successor of s_{i-1} and $\Psi = \{\mathit{eff}_{a_{i-1,j}} | a_{i-1,j} \in A_{i-1}\}$*

The notion of the relaxed parallel plan arises from omitting the mutex encodings in the parallel plan encoding. Therefore the existence of a relaxed parallel plan of

length at most h can be encoded using a subformula of the parallel plan encoding:

$$R_{\Pi}^{h} : I_0 \wedge G_h \wedge \bigwedge_{0 \leq t < h} \left(F_A^t \wedge P_A^t \wedge E_A^t \right) \tag{3.1}$$

Corollary 3.2.1. *Relaxed parallel plans are not sequentializable in general.*

Proof. This can be proven through a concise example. Let us consider a slight modification to the previous planning problem.

$\Pi = (\mathcal{V}_{\mathbb{B}} = \emptyset, \mathcal{V}_{\mathbb{Q}} = \{x,y\}, A, I = \{x = 0, y = 0\}, G = \{x \geq 1, y \geq 1\})$ with
$A = \{a, b\}$ and
$a = (\{y < 1\}, \{x = x + 1\})$
$b = (\{x < 1\}, \{y = y + 1\})$

Here $\pi = \langle \{a,b\} \rangle$ is a relaxed parallel plan, as for $s_0(x) := 0, s_0(y) := 0$ and $s_1(x) := 1, s_1(y) := 1$ holds $s_0 \models I, s_1 \models G$ and $s_0 \models \{y < 1, x < 1\}$. All possible sequentializations $\langle a,b \rangle, \langle b,a \rangle$ however do not yield a valid sequential plan, as the effects and preconditions of the two actions a and b exclude each other, given the initial state I. □

Having established that relaxed parallel plans are not generally sequentializable, let alone trivially, we aim at designing an algorithm, which looks for relaxed parallel plans of increasing horizon and sequentializes a found relaxed parallel plan, if possible. The procedure derives a refinement of the initial encoding, in case the relaxed parallel plan cannot be sequentialized. For this purpose we introduce several adjustable concepts, which mostly have orthogonal roles in the overall algorithm. The overall algorithm is based on the previously introduced incremental solving approach and is summarized in Algorithm 3. Here R is an initially empty set, containing information for refinements of the plain relaxed parallel plan encoding. These are learnt during the search. The exact definition of R depends on the selectable options of the algorithm, as will be described in detail in the following.

In order to explore different refinement procedures, the algorithm possesses these three mostly orthogonal settings and options:

1. *Sequentializability check: general, fixed order, syntactical.*
   This setting defines which method will be used to determine whether a relaxed parallel plan is seqentializable in a certain way.

2. *Unsat core: on, off*
   This setting defines, whether the unsat core of the underlying z3 solver is used.

3. *Timesteps: all, current, dynamic*
   This setting defines for which steps the learnt exclusions are encoded in the refinement process.

Before describing the adjustable parts of the algorithm in greater detail, we give an intuitive overview of the procedures.

Analogously to the previously introduced methods, the existence of a relaxed parallel plan for some planning problem is encoded for increasing horizons. If the encoding is satisfiable for some horizon, a sequentializability check is applied to one

Algorithm 3 Relaxed Parallel Search.

```
Input: Π = (V_B, V_Q, A, I, G), ub
 1: h := 1, R := ∅
 2: initial_s := I_0
 3: solver = z3.Solver()
 4: solver.add(initial_s)
 5: solver.push()
 6: while h ≤ ub do
 7:   refinement = encode_refinement(R,h)
 8:   relaxed_encoding = F^{h-1} ∧ P_A^{h-1} ∧ E_A^{h-1}
 9:   solver.add(encoding, refinement)
10:   solver.push()
11:   solver.add(G_h)
12:   res = solver.check()
13:   while res == sat do
14:     if sequentializable() then
15:       return  sequentialize_plan()
16:     else
17:       solver.pop()
18:       update(R)
19:       refinement = encode_refinement(R,h)
20:       solver.add(refinement)
21:       solver.push()
22:       solver.add(G_h)
23:       res = solver.check()
24:     end if
25:   end while
26:   solver.pop()
27:   h += 1
28: end while
29: return  'No plan found within bound.'
```

relaxed parallel step after the other to determine, whether it is possible to derive a sequential plan. In case the the sequentializability check returns `true`, the check also provides the necessary information to construct the sought sequential plan. If the sequentializability check returns `false`, the check yields information for the subsequent refinement of the relaxed parallel plan encoding. The superordinate setting is the sequentializability check. Depending on the choice of the sequentializability check, the other settings may come into play.

The *syntactical* sequentializability check determines for each relaxed parallel step, whether the set of actions in that step is independent. If it is not, the refinement consists of encoding the corresponding mutexes. With the *Timesteps* option *all*, the mutexes will be encoded for all steps. Analogously, with *Timesteps* set to *current*, the mutexes will be encoded only for the step in which the non-independent relaxed parallel plan step occurred. The other settings can not be applied under this sequentializability check.

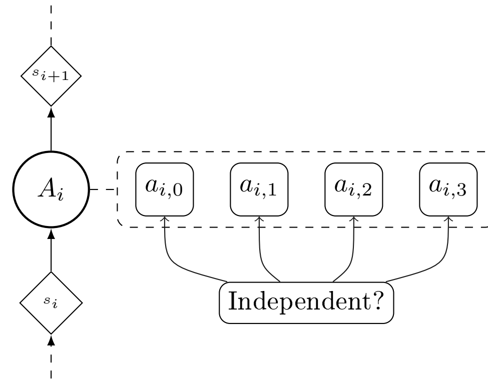

Figure 3.2: Illustration of the syntactical sequentializability check.

The *general* sequentializability check, determines for each relaxed parallel step whether there is any sequential plan with actions from the step, corresponding to the relaxed parallel step. With the *Unsat core* option *off*, the subsequent refinement consists of encoding that the actions in the relaxed parallel step cannot simultaneously appear in any step, if *Timesteps* is set to *all*, or only the corresponding step, otherwise. If *Unsat core* is set to *on*, only conflicting actions will be excluded.

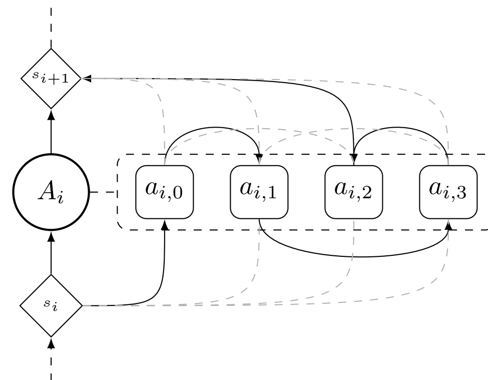

Figure 3.3: Illustration of the general sequentializiability check.

The *fixed order* sequentializability check, assumes a fixed order of all actions. It then determines for each relaxed parallel step whether the actions in that step can be executed in the set order from any state. If that is the case, it is subsequently checked, whether the ordered sequence of actions corresponds to the relaxed parallel step. The following refinement also consists of encoding the exclusion of actions in a relaxed parallel step. Here either all actions or only a conflicting subset are excluded for all or only one step, depending on the *Unsat core* and *Timesteps* settings. Additionally, if *Timesteps* is set to *dynamic*, the exclusion of the set of actions is encoded for all steps, if the first check fails, and for only the corresponding step, if only the second check fails.

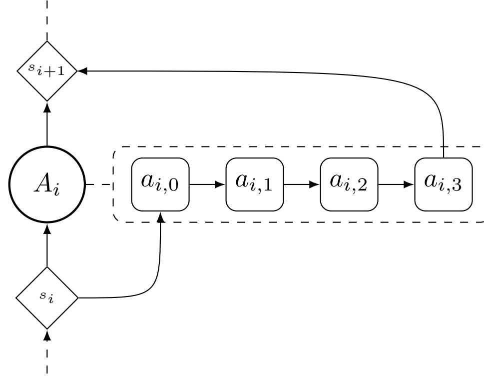

Figure 3.4: Illustration of the fixed order sequentializability check.

In the following we explain the different methods for testing sequentializability and how the settings interact in detail.

3.3 Methods for Testing Sequentializability

We developed three different methods for checking the sequentializability of a relaxed parallel plan. With solving the formula as noted in Line 12 and 23 of Algorithm 3 the solver generates a model J of the formula, if the formula is satisfiable. The `sequentializabe()` routine called in Line 15 of Algorithm 3 consists of subsequently calling sequentializability checks for each relaxed parallel plan step until the first check returns false or all checks succeed. The model J is passed on to each sequentializability check. In case the encoding is satisfiable, the corresponding relaxed parallel plan is $\langle A_0, ..., A_{n-1} \rangle$, with $A_i = \{a \in A | [\![a^t]\!]_J = \top\}$. Be $\langle s_0, ..., s_n \rangle$ the unique sequence of states corresponding to the found relaxed parallel plan. This sequence of states can be directly deduced from the model J: $s_i(v) := [\![v^i]\!]_J$ for $v \in \mathcal{V}$. Relaxed parallel plan steps containing only one action are trivially sequentializable and are not passed to any of the methods.

Definition 3.3.1 (Align)**.** *Given a sequence of actions $\langle a_0, ..., a_{k-1} \rangle$ and a sequence of states $\langle s_0, ..., s_k \rangle$, we say that these sequences* align*, if and only if*

- *each action $a_j, 0 \leq j < k$ is applicable in state s_j and*
- *s_j is the successor of s_{j-1} and the effect of a_{j-1}, for $0 < j \leq k$.*

3.3.1 General Sequentializability Check

The *general sequentializability check* examines for every step A_i, $0 \leq i < n$, whether there is some sequence of actions $\langle a_0, ..., a_{k_i - 1} \rangle$, with $a_j \in A_i, 0 \leq j < k_i$ and $k_i \leq |A_i|$, with a unique sequence of states $\langle s^i_0, ..., s^i_{k_i} \rangle$, such that

- $s^i_0 = s_i$
- $s^i_{k_i} = s_{i+1}$
- $\langle a_0, ..., a_{k_i - 1} \rangle$ and $\langle s^i_0, ..., s^i_{k_i} \rangle$ align

This translates to the planning problem Π^i with an upper bound $ub = |A_i|$.

$$\Pi^i = (\mathcal{V}_{\mathbb{B}}, \mathcal{V}_{\mathbb{Q}}, A_i, I^i, G^i) \text{ ,where} \tag{3.2}$$
$$I^i = \{v = [\![v^i]\!]_J | v \in \mathcal{V}\} \tag{3.3}$$
$$G^i = \{v = [\![v^{i+1}]\!]_J | v \in \mathcal{V}\} \tag{3.4}$$

The following formula prohibits the simultaneous execution of two actions in one step:

$$L_A^t : \bigwedge_{a_1, a_2 \in A, a_1, \neq a_2} (\neg a_1^t \vee \neg a_2^t) \tag{3.5}$$

With this, one can encode the existence of a sequential plan of length at most $|A_i|$ for the planning problem Π^i:

$$S_{\Pi^i} : I_0^i \wedge G_{|A_i|}^i \wedge \bigwedge_{0 \leq t < |A_i|} \left(F_{A^i}^t \wedge P_{A^i}^t \wedge E_{A^i}^t \wedge L_{A^i}^t\right) \tag{3.6}$$

In essence, the sequentializability check encodes the existence of a sequential plan of length at most $|A_i|$ of the planning problem Π^i for each $0 \leq i < n$. If all formulas are satisfiable, the plan is recognized as sequentializable. It is therefore possible to create a sequential plan for the original planning problem by concatenating the sequential plans for each of the Π^i planning problems:

$$\pi_n = \langle s_0^0, ..., s_{k_0}^0, ..., s_0^{n-1}, ..., s_{k_{n-1}}^{n-1} \rangle \tag{3.7}$$

It should be noted that this method does not only check, whether the actions in each relaxed parallel step can be ordered suitably, but also allows to repeat and omit actions as long as the corresponding plan does not exceed the given maximal length.

Abstraction Refinement In case a sequentializability check does not recognize a relaxed parallel plan step as sequentializable, we want to refine our abstract relaxed parallel plan. The refinement should at least exclude the same non-sequentializable solution and optimally should exclude as many other non-sequentializable solutions of the abstraction as possible. This is where the settings 2. and 3. play a decisive role.

The procedure is formally summarized in Algorithm 4, with $\Pi = (\mathcal{V}_{\mathbb{B}}, \mathcal{V}_{\mathbb{Q}}, A, I, G)$ being the planning problem and A_i a relaxed parallel plan step.

Algorithm 4 General Sequentializability Check.

Input: Π, A_i
1: encoding = S_{Π^i}
2: refinement_solver.add(encoding)
3: res = refinement_solver.check()
4: **if** res == sat **then**
5: model = refinement_solver.model()
6: **return** True, plan(model)
7: **end if**
8: **return** False, None

If the sequentializability check returns `False` for the relaxed parallel plan step A_i and *Unsat core* is set to *off*, we refine by excluding the set of actions A_i from appearing in one relaxed parallel step. This means that the `update(`R`)` function in Algorithm 3 is defined as in Algorithm 5.

Algorithm 5 update(R), *Sequentializability check: General, Unsat core: off*

Input: R, A_i
1: $R = R \cup \{A_i\}$

Now, when setting *Unsat core* to *on* we intend to make use of the conflict analysis provided by the z3 solver instance. To do this, each formula, which is part of the encoding of the semantics of executing one specific action (i.e. P_a^t and E_a^t), is asserted to the solver together with a label. This label allows the attribution of the formula to the corresponding unique action. With the unsat core activated, the solver returns a subset of the labels UC, which indicates the subset of formulas involved in the conflict that led to the `unsat` result. Due to the fact that we created labels for the encoding of each action at each step, we can use the unsat core UC to derive a potentially more precise refinement, by excluding the set of actions, which corresponds to the unsat core. This is done by defining the `update(`R`)` function as done in Algorithm 6.

Algorithm 6 update(R), *Sequentializability check: general, Unsat core: on*

Input: R, A_i, UC
1: $R = R \cup \{a \in A_i \mid a \text{ is in } UC\}$

Finally, with the *Timestep* setting we can specify for which steps the refinement in R should be encoded. If *Timestep* is set to *all* the `encode_refinement(`R`)` method from Algorithm 3 behaves such that the following formula is asserted to the solver:

$$\bigwedge_{A_i \in R} \left(\bigwedge_{0 \leq t < h} \neg \left(\bigwedge_{a \in A_i} a^t \right) \right) \tag{3.8}$$

Analogously, if *Timestep* is set to *Current* the `encode_refinement(`R`)` method behaves such that the following formula is asserted to the solver:

$$\bigwedge_{A_i \in R} \neg \left(\bigwedge_{a \in A_i} a^i \right) \tag{3.9}$$

We refer to each subformula consisting of the negation of a conjunction of a set of action variables for some timestep t derived from an unsequentializable relaxed parallel plan step as an *invariant encoding*.

The *dynamic* option allows for the seqentializiability check to decide whether a given refinement should be encoded for all timesteps or only the timestep it occurred in. However, we only make use of this feature with the fixed order sequentializability check.

Example 3.3.1. We illustrate the general sequentializability check in the relaxed parallel search through a small example. Let us assume the following planning problem.

$$\begin{aligned}
\Pi &= (\mathcal{V}_{\mathbb{B}} = \emptyset, \mathcal{V}_{\mathbb{Q}} = \{x,y,z\}, A, I, G)\\
I &= \{x = 0, y = 0, z = 0\}\\
G &= \{x \geq 1, y \geq 1, z \geq 1\}\\
A &= \{a, b, c, d\} \text{ and}\\
a &= (\{y < 1\},\{x = x + 1\})\\
b &= (\{x < 1\},\{y = y + 1\})\\
c &= (\{x \geq 1\},\{y = y + 1\})\\
d &= (\{x \leq 0\},\{z = z + 1\})
\end{aligned}$$

Now, applying the relaxed parallel search method with *Seq. check: general, Unsat core: off, Timesteps: all* to this problem would start by encoding the formula R^1_Π. This formula has a unique model yielding the unique relaxed parallel plan $\langle A_1 = \{a,b,d\}\rangle$. To test the sequentializability of the plan, the formula S_{Π^1} is encoded. This formula is unsatisfiable as the actions a and b can not be executed subsequently in any order given the initial state. Subsequently the previous relaxed parallel plan encoding is refined to:

$$R^1_\Pi \wedge \neg(a^1 \wedge b^1 \wedge d^1)$$

The refined formula is unsatisfiable, because the refinement excludes the only model of R^1_Π. Therefore the horizon is incremented and a formula expressing the existence of a longer relaxed parallel plan is encoded. Due to the *Timestep* option being set to *all* the refinement encoding is amended correspondingly:

$$R^2_\Pi \wedge \neg(a^1 \wedge b^1 \wedge d^1) \wedge \neg(a^2 \wedge b^2 \wedge d^2)$$

This formula has several models. For simplicity let us assume that the solver finds the model with the corresponding relaxed parallel plan $\langle A_1 = \{a,d\}, A_2 = \{c\}\rangle$. During the sequentializability check the corresponding formula S_{Π^1} is encoded. It has a unique model ordering the actions in A_1 to $\langle d, a\rangle$. The relaxed parallel plan step A_2 is recognized to be trivially sequentializable due to containing only a single action. Consequently the relaxed parallel search terminates, returning the sequential plan $\langle d, a, c\rangle$.

In the (not relaxed) parallel search given the same planning problem mutexes for each pair of actions from A are encoded, as the variable x appears in either the precondition or effect of each action. Consequently the encodings P^h_Π for $h = 1$ and $h = 2$ are unsatisfiable and a plan is only found at horizon $h = 3$.

Lemma 3.3.1. *Given a planning problem $\Pi = (\mathcal{V}_{\mathbb{B}}, \mathcal{V}_{\mathbb{Q}}, A, I, G)$, two states s_i,s_{i+1} and a subset $A_i \subseteq A$, if the general sequentializability check, returns a sequence of actions $\langle a_0,...,a_{k_i-1}\rangle$, with $a_j \in A_i, 0 \leq j < k_i$ and $k_i \leq |A_i|$, then there is a sequence of states $\langle s^i_0,...,s^i_{k_i}\rangle$, such that $s^i_0 = s_i$, $s^i_{k_i} = s_{i+1}$ and $\langle a_0,...,a_{k_i-1}\rangle$ and $\langle s^i_0,...,s^i_{k_i}\rangle$ align .*

Proof. As described above, based on the input the existence of a sequential plan of a length less of equal to $|A_i|$ for the planning problem Π^i is encoded. Given the correctness of the encoding of the existence of a sequential plan, the lemma follows. □

It should be noted that it also holds that, if there is a sequence of states $\langle s^i_0,...,s^i_{k_i}\rangle$, such that $s^i_0 = s_i$, $s^i_{k_i} = s_{i+1}$ and $\langle a_0,...,a_{k_i-1}\rangle$ and $\langle s^i_0,...,s^i_{k_i}\rangle$ align, the general sequentializability check returns some sequence of actions from A_i. However, the returned sequence of actions is not unique and for our purposes it suffices to show Lemma 3.3.1.

3.3.2 Fixed Order Sequentializability Check

For the *fixed order sequentializibility check* initially a total ordering $<$ of the actions in A is set. In the current implementation this order is picked without any particular strategy. The method can be executed in two stages for every step A_i, $0 \leq i < n$.

The *global fixed order sequentializability check* examines, whether there is a sequence $\rho_{A_i} = \langle s^i_0,...,s^i_{|A_i|}\rangle$ of states, such that ρ_{A_i} and the sequence of actions resulting from the ordering of A_i according to $<$ align. The objective of this is to determine whether the actions may ever occur in one step.

If we use $\sigma^<_{A_i} : A_i \to \{0,...,|A_i| - 1\}$ to assign to each action its positions after one would order A_i, we can encode the problem of the existence of said sequence into the following formula. For the sake of readability we abbreviate $\sigma^<_{A_i}$ trough σ

$$O^g_{A_i} : \bigwedge_{a \in A_i} (a^{\sigma(a)} \wedge P^{\sigma(a)}_a \wedge E^{\sigma(a)}_a \wedge F^{\sigma(a)}_{\{a\}}) \tag{3.10}$$

The *local fixed order sequentializability check* adds the requirement, that $s^i_0 = s_i$ and $s^i_{k_i} = s_{i+1}$. This is necessary to determine whether the actions in A_i ordered according to $<$ form a suitable sequence of actions given a concrete initial and final state. Conveniently it suffices to amend the previous formula to encode the additional requirement.

$$O^l_{A_i} : O^g_{A_i} \wedge \bigwedge_{v \in \mathcal{V}_{\mathbb{Q}}} (v_0 = s_i(v) \wedge v_{|A_i|} = s_{i+1}(v)) \tag{3.11}$$

$$\wedge \bigwedge_{v \in \mathcal{V}_{\mathbb{Q}}, s_i(v) = \top} v_0 \wedge \bigwedge_{v \in \mathcal{V}_{\mathbb{Q}}, s_{i+1}(v) = \top} v_{|A_i|} \tag{3.12}$$

$$\wedge \bigwedge_{v \in \mathcal{V}_{\mathbb{Q}}, s_i(v) = \bot} \neg v_0 \wedge \bigwedge_{v \in \mathcal{V}_{\mathbb{Q}}, s_{i+1}(v) = \bot} \neg v_{|A_i|} \tag{3.13}$$

Abstraction Refinement The usage of the unsat core and the corresponding refinement is analogous to the general sequentializability check. Also the *all, current* options for the *Timesteps* setting are exactly analogue. Additionally we have the

dynamic setting, which distinguishes between the local and the global fixed order sequentializability check. If for the relaxed parallel step already the global check returns `unsat`, the refinement is encoded for all steps:

$$\bigwedge_{0 \leq t < h} \neg \left(\bigwedge_{a \in A_i} a^t \right) \tag{3.14}$$

The reasoning being that in this case it is implied that there is no sequence of states ρ, such that ρ and the sequence of ordered actions from A_i align. Accordingly, if the global check returns `sat` and the local check returns `unsat` the refinement is encoded only for the corresponding step:

$$\neg \left(\bigwedge_{a \in A_i} a^i \right) \tag{3.15}$$

Example 3.3.2. For the purpose of illustration we will view another concise example. Let us solve the following planning problem with the relaxed parallel search method and *Seq. check: fixed order, Unsat core: off, Timesteps: dynamic*:

$$\begin{aligned}
&\Pi = (\mathcal{V}_{\mathbb{B}} = \{x_1\}, \mathcal{V}_{\mathbb{Q}} = \{y_1, y_2, y_3\}, A, I, G)\\
&I = \{x_1 = \bot, y_1 = 0, y_2 = 0, y_3 = 0\}\\
&G = \{y_1 > 0, y_2 > 0, y_3 \geq 1\}\\
&A = \{a, b, c, d\} \text{ and}\\
&\quad a = (\{\neg x_1\}, \{x_1, y_1 = y_1 + 1\})\\
&\quad b = (\{\neg x_1\}, \{x_1, y_2 = y_2 + 1, y_3 = y_3 + 1\})\\
&\quad c = (\{x_1\}, \{y_2 = y_2 + 1\})\\
&\quad d = (\{x_1, y_2 = 0\}, \{y_3 = y_3 + 1\})
\end{aligned}$$

We assume a lexicographical ordering of the actions.

Initially the relaxed parallel plan for horizon $h = 1$ is encoded R^1_Π. The formula has a unique model, giving the relaxed parallel plan $\langle A_0 = \{a, b\}\rangle$. The corresponding formula $O^g_{A_0}$ of the global fixed order sequentializability check is unsatisfiable, as action a deactivates the precondition of action b. Therefore the refinement $\neg(a \wedge b)$ will be encoded for every step. The refined relaxed parallel plan encoding $R^1_\Pi \wedge (\neg(a^0 \wedge b^0))$ is unsatisfiable. Thus, the formula is encoded for the subsequent horizon:

$$R^2_\Pi \wedge (\neg(a^0 \wedge b^0)) \wedge (\neg(a^1 \wedge b^1)) \tag{3.16}$$

This yields the relaxed parallel plan $\langle A_0 = \{a\}, A_1 = \{c, d\}\rangle$. Now the global sequentializability check for A_1 passes, but the local check fails. This is due to the fact that executing c directly before d enables the precondition of d if and only if $y_2 = -1$. Consequently the refinement $\neg(c \wedge d)$ is only encoded for the step $t = 1$:

$$R^2_\Pi \wedge (\neg(a^0 \wedge b^0)) \wedge (\neg(a^1 \wedge b^1)) \wedge (\neg(c^1 \wedge d^1)) \tag{3.17}$$

This formula is unsatisfiable leading us to the the encoding for horizon $h = 3$:

$$R^3_\Pi \wedge (\neg(a^0 \wedge b^0)) \wedge (\neg(a^1 \wedge b^1)) \wedge (\neg(c^1 \wedge d^1)) \wedge (\neg(a^2 \wedge b^2)) \tag{3.18}$$

The above encoding has multiple models. For the sake of brevity, we assume that the solver finds the solution which yields the relaxed parallel plan $\langle A_0 = \{a\}, A_1 = \{d\}, A_2 = \{c\}\rangle$, directly resulting in finding the sequential plan $\langle a,d,c\rangle$.

Lemma 3.3.2. *Given a planning problem* $\Pi = (\mathcal{V}_\mathbb{B}, \mathcal{V}_\mathbb{Q}, A, I, G)$, *two states* s_i,s_{i+1} *and a subset* $A_i \subset A$, *if the fixed order sequentializability check, returns a sequence of actions* $\langle a_0,\ldots,a_{k_i-1}\rangle$, *with* $a_j \in A_i, 0 \leq j < k_i$ *and* $k_i = |A_i|$, *then there is a sequence of states* $\langle s^i_0,\ldots,s^i_{k_i}\rangle$, *such that* $s^i_0 = s_i$, $s^i_{k_i} = s_{i+1}$ *and* $\langle a_0,\ldots,a_{k_i-1}\rangle$ *and* $\langle s^i_0,\ldots,s^i_{k_i}\rangle$ *align.*

Proof. Given the correctness of the encoding for the local fixed order sequentializability check, we know that the check returns a sequence of states $\langle a_0,\ldots,a_k\rangle$, only if $a_j = a' \Leftrightarrow \sigma(a') = j$, $k = |A_i|$ and there is a sequence of states meeting the requirements stated above. □

3.3.3 Syntactical Sequentializability Check

The relaxed parallel search with the *syntactical sequentializability check*, see Algorithm 7, corresponds to parallel search with a reactive mutex encoding. Instead of encoding all syntactically derived mutexes from the start, the set of actions in each relaxed parallel step is checked for independence with the same criteria, as for the mutexes in the parallel search.

Algorithm 7 Syntactical Sequentializability Check

Input: A_i
1: mutexes = computeMutexes(A_i)
2: **if** mutexes == $\emptyset$ **then**
3:    **return** True, tuple(A_i)
4: **end if**
5: **return** False, None

Only in case a non-independent set is found, the mutex excluding this step, will be encoded, see Algorithm 8.

Algorithm 8 update(R), *Sequentializability check: syntactical*

Input: R, A_i
1: mutexes = computeMutexes(A_i)
2: $R = R \cup \{\text{mutexes}\}$

The *all, current* options for the *Timesteps* setting are analogue to the previous methods. As this method is restricted to the syntactical analysis of abstract actions, all other settings do not find any application.

Example 3.3.3. To illustrate this method in comparison to the parallel search we consider the following planning problem and *Timesteps: current*.

$$\begin{aligned}
&\Pi = (\mathcal{V}_{\mathbb{B}} = \emptyset, \mathcal{V}_{\mathbb{Q}} = \{y_1,y_2,y_3\}, A, I, G)\\
&I = \{x = 0, y = 0, z = 0\}\\
&G = \{x \geq 1, y \geq 1, z \geq 1\}\\
&A = \{a, b, c\} \text{ and}\\
&\quad a = (\{y = 0\},\{z = z + 1\})\\
&\quad b = (\{y \leq 0\},\{x = x + 1, y = y + 1\})\\
&\quad c = (\{y > 0\},\{z = z + 1\})
\end{aligned}$$

The initial encoding R^1_Π has a unique solution with the relaxed parallel plan $\langle A_0 = \{a,b\}\rangle$. The sequentializability check finds the first step not to be sequentializable due to the variable y occurring in the precondition of a and an effect of b. Subsequently the encoding is refined to $R^1_\Pi \wedge \neg(a^0 \wedge b^0)$, which is unsatisfiable. Finally, a unique model is found for the encoding with the next horizon $R^2_\Pi \wedge \neg(a^0 \wedge b^0)$. This results in the relaxed parallel plan $\langle A_0 = \{b\}, A_1 = \{c\}\rangle$ from which the sequential plan $\langle b, c\rangle$ is directly derived. Solving the same planning problem with the parallel search would make it necessary to encode mutexes for each pair of actions for all steps. Therefore the final encoding would correspond to:

$$R^1_\Pi \wedge \neg(a^0 \wedge b^0) \wedge \neg(a^0 \wedge c^0) \wedge \neg(b^0 \wedge c^0) \tag{3.19}$$

$$\wedge \neg(a^1 \wedge b^1) \wedge \neg(a^1 \wedge c^1) \wedge \neg(b^1 \wedge c^1) \tag{3.20}$$

Lemma 3.3.3. *Given a planning problem $\Pi = (\mathcal{V}_{\mathbb{B}}, \mathcal{V}_{\mathbb{Q}}, A, I, G)$, two states s_i,s_{i+1} and a subset $A_i \subset A$, if the syntactical sequentializability check, returns a sequence of actions $\langle a_0,...,a_{k_i-1}\rangle$, with $a_j \in A_i, 0 \leq j < k_i$ and $k_i = |A_i|$, then there is a sequence of states $\langle s^i_0,...,s^i_{k_i}\rangle$, such that $s^i_0 = s_i$, $s^i_{k_i} = s_{i+1}$ and $\langle a_0,...,a_{k_i-1}\rangle$ and $\langle s^i_0,...,s^i_{k_i}\rangle$ align.*

Proof. Due to requiring independence of the set of actions A_i, the syntactical sequentializability check, returns the sequence of actions $\langle a_0,...,a_{k_{i-1}}\rangle$ only if, $\langle A_i\rangle$ is a relaxed parallel plan for the planning problem $\Pi' = (\mathcal{V}_{\mathbb{B}}, \mathcal{V}_{\mathbb{Q}}, A, I^i = \{v = [\![v]\!]_{s_i} | v \in \mathcal{V}\}$, $G^i = \{v = [\![v]\!]_{s_{i+1}} | v \in \mathcal{V}\})$. Assuming the correctness of the notion of a parallel plan the sequence $\langle a_0,...,a_{k_{i-1}}\rangle$ suffices said conditions. □

3.3.4 Soundness and Completeness.

Theorem 3.3.4. *The relaxed parallel search is sound.*

Proof. Be $\langle a_0,...,a_{j-1}\rangle$ a plan found for some numeric planning problem $\Pi = (\mathcal{V}_{\mathbb{B}}, \mathcal{V}_{\mathbb{Q}}, A, I, G)$. Following Lemmata 3.3.2, 3.3.1 and 3.3.3, we can deduct that the plan can be split into $i \leq j$ subsequences $\rho_k = \langle a_{k,0},...,a_{k,l_k-1}\rangle, 0 \leq k < i$, such that there are i sequences of states $\eta_k = \langle s_{k,0},...,s_{k,l_k}\rangle, 0 \leq k < i$ where for each $0 \leq k < i$ it holds that ρ_k and η_k align and for $k < i - 1$ it holds that $s_{k,l_k} = s_{k+1,0}$. Through the encoding it is ensured that $s_{0,0} \models I$ and $s_{i-1,l_{i-1}} \models G$. Hence $\langle a_0,...,a_{j-1}\rangle$ is a valid plan for the planning problem Π. □

Theorem 3.3.5. *The relaxed parallel search is complete.*

Proof. Be $\Pi = (\mathcal{V}_{\mathbb{B}}, \mathcal{V}_{\mathbb{Q}}, A, I, G)$ any numeric planning problem, which has a solution in form of a plan $\langle a_0,\ldots,a_{j-1}\rangle$.

Given the correctness of the encoding of the existence of a sequential plan with length less or equal to j, there is a model J, such that $J \models R^j_\Pi \wedge \bigwedge_{0\leq t<h} L^t_A$.

At each horizon in the relaxed parallel search there are only finitely many iterations of refinement, as each refinement excludes the previous model and does not allow for models, which were not models of the formula prior to the refinement. Therefore the relaxed parallel search either finds some correct plan at a horizon smaller then j or reaches the horizon j.

The formula in the relaxed parallel search at horizon j is the conjunction of R^j_Π and the encoding of some refinement Enc_{ref}. As $J \models R^j_\Pi \wedge \bigwedge_{0\leq t<h} L^t_A$ (encoding of a sequential plan) it is implied that $J \models R^j_\Pi$ and $J \models Enc_{ref}$, because while $\bigwedge_{0\leq t<h} L^t_A$ excludes the execution of more then one action at any step, Enc_{ref} only excludes the simultaneous execution of some sets of actions with at least two actions for some steps. Consequently the relaxed parallel search finds a valid plan for Π at the latest when reaching horizon h. □

Chapter 4

Experimental Results

We evaluate an implementation extending OMTPlan on standard benchmarks for numeric planning problems taken from the literature. All experiments were executed on the same machine running under Debian and possessing 192GB of memory and four 12-core AMD Opteron 6172 processors each with a frequency of 2.1GHz. All of the search methods were single threaded.

We ran several small tests to observe the behaviour of our relaxed parallel search under different settings. The universally best setting turned out to be:

1. *Sequentializability check: general*
2. *Unsat core: on*
3. *Timesteps: all*

In the following the relaxed parallel search with the above settings will be referred to as *CEGAR*.

4.1 Comparisons

We compared the *CEGAR* method and the incremental parallel search to both the parallel search and the state of the art planning software *springroll* [SRHT16] on a broader set of instances with a timeout of 30 minutes. In the following plots *VBS* represents the virtual best solver with regards to the other planning algorithms. The resulting solving times for domains containing only simple conditions can be seen in Table 4.1 and for domains also containing linear conditions in Table 4.2.

4.2 Analysis

The overall highest coverage is achieved by springroll. Especially on domains where long plans are needed, the encoding used in springroll has a clear advantage. The incremental parallel search poses a clear improvement of the parallel search on many instances and has the overall best performance on some domains.

Domain	#	CEGAR		par. incr.		springroll		parallel		VBS	
		C	T	C	T	C	T	C	T	C	T
fn-counters-inv	10	9	2696	**10**	2295	**10**	10	**10**	1282	10	9
fn-counters	10	**10**	920	**10**	497	**10**	9	**10**	911	10	8
satellite	20	**6**	1039	5	2072	3	324	5	1993	7	2197
depots numeric	20	3	1277	**5**	2934	4	1728	3	2154	6	4119
sailing	20	6	1457	4	15	0	0	**11**	6699	11	6667
farmland	20	0	0	0	0	**20**	19	0	0	20	19
fn-counters-rnd	20	**20**	330	**20**	182	**20**	17	**20**	317	20	13
gardening	20	12	1739	12	1537	**20**	31	14	1225	20	29
rover-numeric	20	12	3324	**13**	2108	11	2382	12	5416	13	2073
zeno-travel-small	10	**10**	1076	9	2086	*	*	**10**	2170	10	1076
Total	170	88	13857	88	13726	**98**	4521	95	22168	127	16209

Table 4.1: Coverage (C) and total solving time (T) in seconds for domains with simple conditions. The number of tested instances per domain is noted in the (#)-column. *Springroll crashed while parsing instances of the zeno-travel-small domain.

Domain	#	CEGAR		par. incr.		springroll		parallel		VBS	
		C	T	C	T	C	T	C	T	C	T
zeno-travel-linear	10	8	102	9	1104	*	*	**10**	2271	10	1644
fo counters rnd	20	**20**	23	**20**	12	**20**	19	**20**	23	20	11
fo counters inv	20	**20**	1803	18	805	**20**	30	**20**	566	20	27
sailing ln	20	**1**	20	**1**	60	0	0	**1**	39	1	20
farmland ln	20	4	1048	2	1262	**20**	20	3	803	20	20
tpp	20	4	48	4	108	0	0	**5**	1562	5	1422
fo counters	20	**20**	952	**20**	192	**20**	26	**20**	268	20	23
Total	138	85	4068	82	3633	**88**	101	87	6128	104	3171

Table 4.2: Coverage (C) and total solving time (T) in seconds for domains with linear conditions. The number of tested instances per domain is noted in the (#)-column. *Springroll crashed while parsing instances of the zeno-travel-linear domain.

In order to gain a deeper understanding of the behaviour of our abstraction refinement process, we recorded further statistics. To compare the encoding of the CEGAR method to the (not relaxed) parallel search we counted the number of mutex encodings in the encoding of the parallel plan and the number of encoded invariants, see Equation 3.8, which amend the relaxed parallel plan encoding. These statistics are shown for instances of the domain zeno-travel-linear in Table 4.3.

At a closer look, the data suggests that the CEGAR method often does perform significantly better than the parallel search and for some domains better than all other measured search methods for the first few instances of the domain. Unfortunately this advantage does not appear to scale with rising complexity of the problem instances. This can be attributed to multiple causes. Fore one, for smaller problem instances the combinatorial complexity of solving the encoding is negligible in relation to the expense of constructing the encoding. Here, the CEGAR method has an advantage due to its shorter encoding. With rising combinatorial complexity this advantage becomes insignificant. Further, with an increasing horizon the sequentializability checks become more complex as more formulas have to be encoded and solved and more refinement might be needed until a valid plan can be found.

Instance	CEGAR			parallel		
	horizon	invariants	T	horizon	mutexes	T
0	6	12	2	8	4608	14
1	5	0	1	6	3456	8
2	4	4	2	5	9630	18
3	5	35	7	5	12105	22
4	5	35	10	6	27552	54
5	7	182	38	7	38780	84
6	6	102	23	8	52928	126
7	6	18	19	7	161175	392
8	-	-	-	9	234225	730
9	-	-	-	9	263925	824

Table 4.3: Solving time (T) in seconds, horizon and number of encodings of invariants/mutexes of the last encoding for instances of the domain zeno-travel-linear. Timed out instances are marked with '-'.

Table 4.3 very well represents strong points and weak points of our CEGAR approach. From instances 0 to 7 the CEGAR search allows for shorter and far more lightweight encodings resulting in a vastly quicker solving time. This effect however does not appear to be scaleable as for the more complex instances 8 and 9 the refinement process abruptly becomes too complex and the search does not terminate within the given 30 minute timeout.

To analyse this effect in greater detail, we logged the time of each subroutine and visualized them in Figure 4.5 and 4.6. Both figures are bar graphs showing the total solving time of the parallel incremental and CEGAR search for one instance. Each bar is divided into rectangles where the lower end of the rectangle corresponds to the time of the start of a subroutine in the search and the height corresponds to the duration of that subroutine. Each rectangle is annotated with the name of that subroutine, if the height suffices to avoid collisions. In Figure 4.5 we can observe that the CEGAR search has a quicker encoding process and due to only needing one refinement, finds a plan in almost half the time the parallel incremental search needs. In Figure 4.6 we again see that this advantage does not scale well as a more complex instance of the same domain requires a much higher number of refinements resulting in a longer total solving time in relation to the parallel incremental search.

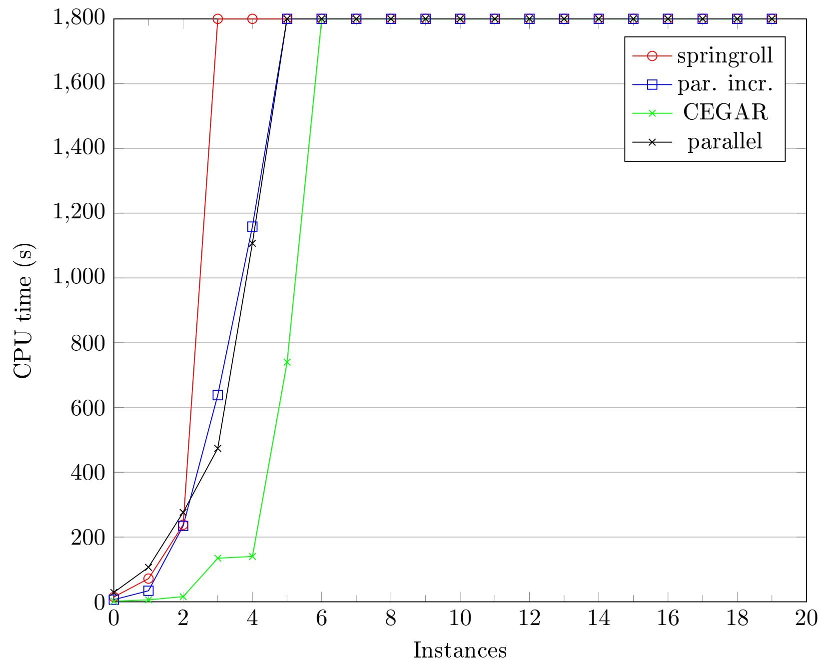

Figure 4.1: Cactus plot for the satellite domain. The instances are ordered by increasing CPU time.

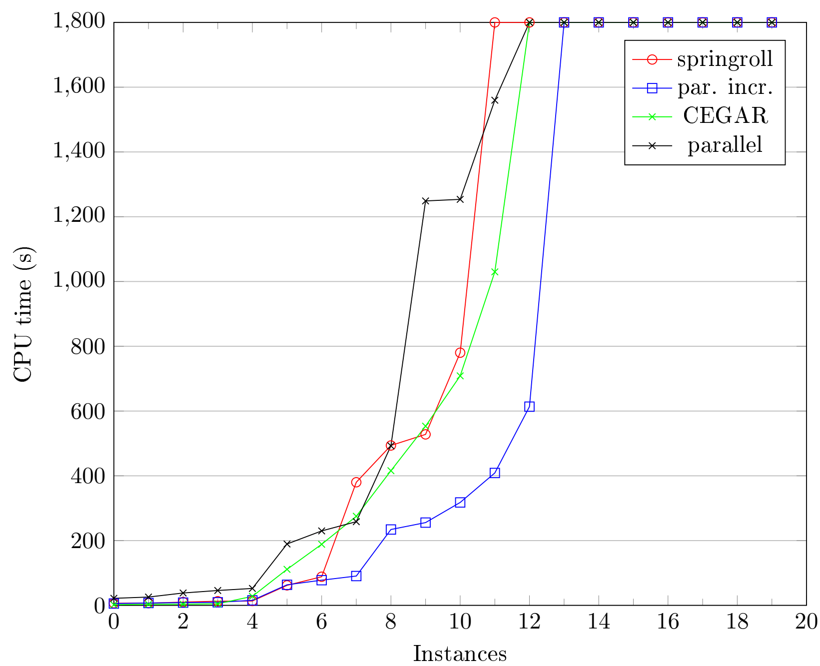

Figure 4.2: Cactus plot for the rover-numeric domain. The instances are ordered by increasing CPU time.

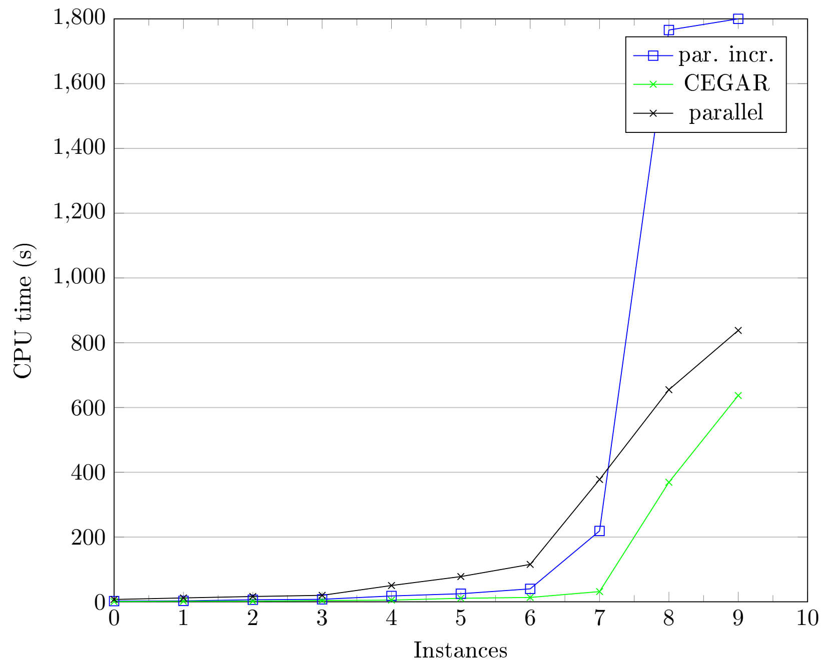

Figure 4.3: Cactus plot for the zeno-travel-small domain. The instances are ordered by increasing CPU time.

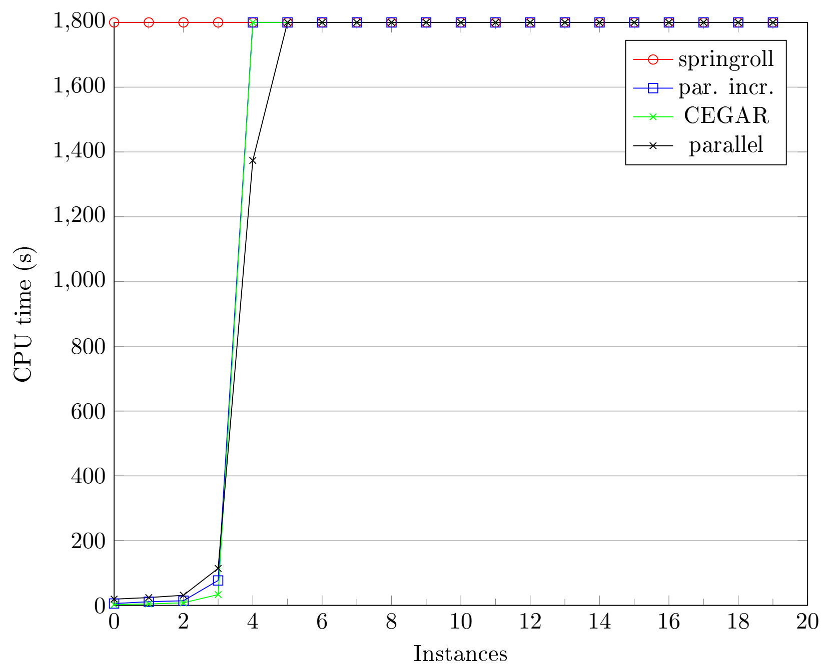

Figure 4.4: Cactus plot for the tpp domain. The instances are ordered by increasing CPU time.

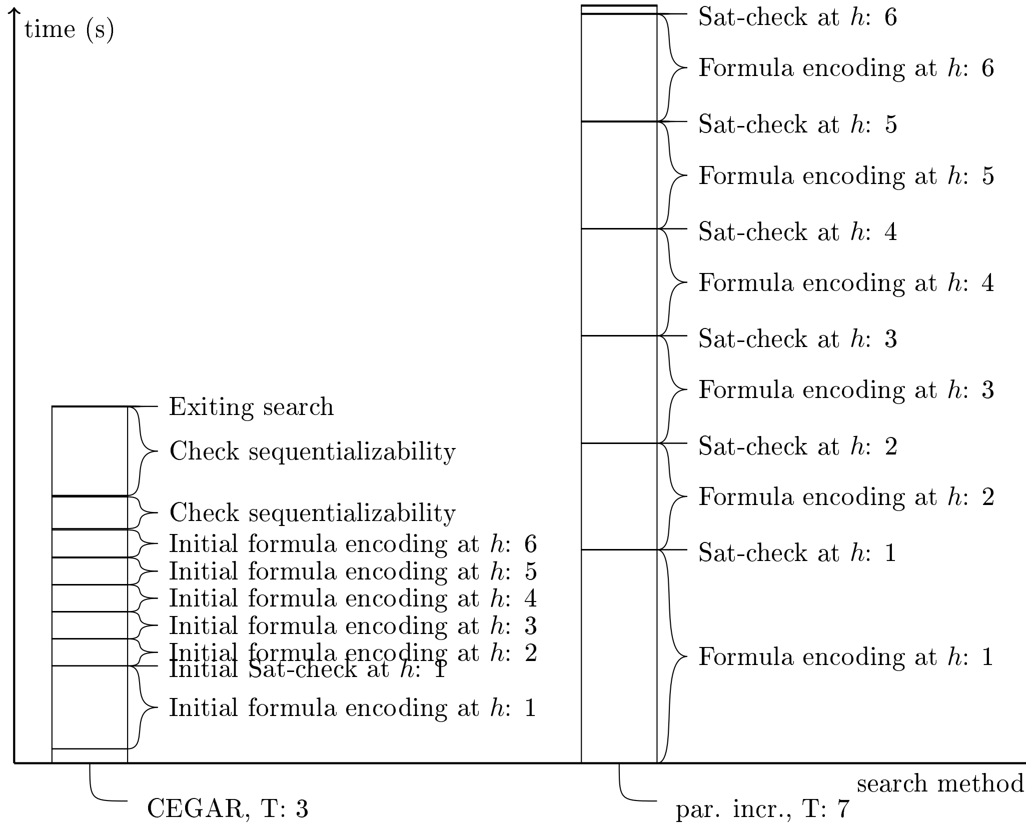

Figure 4.5: Illustration showing the total solving time (T) and duration of the most time consuming subroutines in the parallel and CEGAR search for instance 4 of the rover-numeric domain.

time (s)

Exiting search
Check sequentializability
Check sequentializability
Check sequentializability
Check sequentializability
Check sequentializability
Check sequentializability
Check sequentializability
Check sequentializability
Check sequentializability
Check sequentializability
Check sequentializability
Refined sat-check.
Check sequentializability
Check sequentializability
Check sequentializability
Check sequentializability
Check sequentializability
Check sequentializability
Check sequentializability
Check sequentializability
Check sequentializability
Check sequentializability
Check sequentializability
Check sequentializability
Check sequentializability
Check sequentializability
Refined sat-check.
Check sequentializability
Check sequentializability
Check sequentializability
Check sequentializability
Check sequentializability
Check sequentializability
Check sequentializability
Check sequentializability
Initial formula encoding at h: 4
Initial formula encoding at h: 2
Initial formula encoding at h: 1

Sat-check at h: 6
Formula encoding at h: 6
Sat-check at h: 5
Formula encoding at h: 5
Sat-check at h: 4
Formula encoding at h: 4
Sat-check at h: 3
Formula encoding at h: 3
Sat-check at h: 2
Formula encoding at h: 2
Sat-check at h: 1
Formula encoding at h: 1

search method
CEGAR, T: 73
par. incr., T: 72

Figure 4.6: Illustration showing the total solving time (T) and duration of most time consuming subroutines in the incremental parallel and CEGAR search for instance 12 of the rover-numeric domain.

Chapter 5

Conclusion

5.1 Summary

In this thesis, we discussed an existing SMT-based approach to numeric planning and viewed its encoding in detail. We developed a modified version of that algorithm, which takes advantage of the incremental structure of the encoding.

Further we proposed a new approach of allowing parallelism in the encoding of plans by introducing a relaxation of the original encoding and defining an algorithm, that applies a refinement procedure on this abstraction to find plans. We also introduced multiple methods for some of the subroutines of this algorithm.

With an empirical evaluation we compared our new approaches to both the original algorithm and another state of the art SMT-based planning tool. Our main findings were that our CEGAR method outperforms the other methods on many small problem instances and specific domains, but generally has scalability issues. The incrementality update, despite not scaling well on all domains, generally poses a significant improvement of the original search.

5.2 Future Work

The findings of this thesis immediately yield the basis for two open tasks.

First, since the encoding for the satisficing setting of numeric planning used in OMTPlan is the basis for the encoding used for optimal planning, it would be interesting to investigate whether an incremental solving strategy could be applied there too.

Second, the issue of scalability of our CEGAR approach could be further researched. One could attempt to integrate the CEGAR method into the incremental parallel search by developing a method to switch form the abstraction to the regular mutex encoding after a certain horizon or problem complexity.

Another potentially promising approach, that we left unexplored yet, would be to leverage parallelism to speed up the sequentializability check and refinement process. In our implementation we only used one solver instance for all sequentializability checks in order to speed up the encoding process and take advantage of potential benefits of incremental solving. The opposing approach would be to solve the sequentializability check for each relaxed parallel plan step in parallel. As the input of

each check is fixed after finding a relaxed parallel plan, the checks could directly be executed in parallel. This might not only speed the process up, but could also lead to needing fewer refinement iterations. In the current implementation the sequentializability check terminates after the first not sequentializable relaxed parallel plan step. With a parallel execution of the checks, multiple invariants could be derived in one refinement iteration.

Bibliography

[BdMNW18] Nikolaj Bjørner, Leonardo de Moura, Lev Nachmanson, and Christoph M. Wintersteiger. Programming Z3. In Jonathan P. Bowen, Zhiming Liu, and Zili Zhang, editors, *Engineering Trustworthy Software Systems - 4th International School, SETSS 2018, Chongqing, China, April 7-12, 2018, Tutorial Lectures*, volume 11430 of *Lecture Notes in Computer Science*, pages 148–201. Springer, 2018.

[BEV16] Miquel Bofill, Joan Espasa, and Mateu Villaret. The RANTANPLAN planner: system description. *Knowl. Eng. Rev.*, 31(5):452–464, 2016.

[FL11] Maria Fox and Derek Long. PDDL2.1: an extension to PDDL for expressing temporal planning domains. *CoRR*, abs/1106.4561, 2011.

[LGÁT20] Francesco Leofante, Enrico Giunchiglia, Erika Ábrahám, and Armando Tacchella. Optimal planning modulo theories. In Christian Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, pages 4128–4134. ijcai.org, 2020.

[Rin15] Jussi Rintanen. Models of action concurrency in temporal planning. In Qiang Yang and Michael J. Wooldridge, editors, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 1659–1665. AAAI Press, 2015.

[SD05] Ji-Ae Shin and Ernest Davis. Processes and continuous change in a sat-based planner. *Artif. Intell.*, 166(1-2):194–253, 2005.

[SRHT16] Enrico Scala, Miquel Ramírez, Patrik Haslum, and Sylvie Thiébaux. Numeric planning with disjunctive global constraints via SMT. In Amanda Jane Coles, Andrew Coles, Stefan Edelkamp, Daniele Magazzeni, and Scott Sanner, editors, *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling, ICAPS 2016, London, UK, June 12-17, 2016*, pages 276–284. AAAI Press, 2016.

[SS11] Ashish Sabharwal and Bart Selman. S. russell, p. norvig, artificial intelligence: A modern approach, third edition. *Artif. Intell.*, 175(5-6):935–937, 2011.