Bachelor of Science Thesis

SMT-based Planning for Autonomous Robot Fleets

Leonard Korp

Examiners:
Prof. Dr. Erika Ábrahám
Prof. Dr. Gerhard Lakemeyer

Additional Advisors:
Francesco Leofante
Gereon Kremer

Aachen, 3rd August 2017

Abstract

Industry 4.0 introduces smart factories which are dependent on autonomous production units. One of the most relevant challenges in implementing these factories is to manage and optimize the interactions between the autonomous production units and their environment. Therefore, the present thesis addressed this problem by presenting an approach to generate action plans for the simplified smart factory environment of the RoboCup Logistics League (RCLL). More specifically, a procedure has been derived to create formulas whose models represent realizable action sequences. This property was used to find an optimal action sequence by means of either a plain or an optimizing SMT-solver. Experiments in diverse test environments showed that the crucial factor of the solving times seems to be manageable by this approach. Further research is needed to prove the viability of the presented approach in an actual game of the RoboCup Logistics League.

Eidesstattliche Versicherung

Korp, Leonard	302582
Name, Vorname	Matrikelnummer (freiwillige Angabe)

Ich versichere hiermit an Eides Statt, dass ich die vorliegende ~~Arbeit~~/Bachelorarbeit/ ~~Masterarbeit~~* mit dem Titel

SMT-based Planning for Autonomous Robot Fleets

selbständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Für den Fall, dass die Arbeit zusätzlich auf einem Datenträger eingereicht wird, erkläre ich, dass die schriftliche und die elektronische Form vollständig übereinstimmen. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Aachen, 03.08.2017	
Ort, Datum	Unterschrift

*Nichtzutreffendes bitte streichen

Belehrung:

§ 156 StGB: Falsche Versicherung an Eides Statt

Wer vor einer zur Abnahme einer Versicherung an Eides Statt zuständigen Behörde eine solche Versicherung falsch abgibt oder unter Berufung auf eine solche Versicherung falsch aussagt, wird mit Freiheitsstrafe bis zu drei Jahren oder mit Geldstrafe bestraft.

§ 161 StGB: Fahrlässiger Falscheid; fahrlässige falsche Versicherung an Eides Statt

(1) Wenn eine der in den §§ 154 bis 156 bezeichneten Handlungen aus Fahrlässigkeit begangen worden ist, so tritt Freiheitsstrafe bis zu einem Jahr oder Geldstrafe ein.

(2) Straflosigkeit tritt ein, wenn der Täter die falsche Angabe rechtzeitig berichtigt. Die Vorschriften des § 158 Abs. 2 und 3 gelten entsprechend.

Die vorstehende Belehrung habe ich zur Kenntnis genommen:

Aachen, 03.08.2017	
Ort, Datum	Unterschrift

Contents

1 Introduction

Today's industrial manufacturing is in the process of moving from static process chains to more autonomous and intelligent systems. This paradigm shift is known under the term Industry 4.0. To achieve these goals, so called "smart factories" are established in industrial manufacturing. Smart factories include information processing technology to enable their systems to be autonomous, context-aware and more flexible. One advantage resulting from this is a more dynamic production process, which, instead of having predetermined production steps, can be easily altered to meet changing production demands. Another advantage is the cost reduction if a larger variety of items or small lot sizes have to be produced[KWJ13].

One of the most relevant challenges implementing smart factories is to manage and optimize the interactions between the autonomous production units and their environment. In an attempt to overcome this and other challenges, the RoboCup Logistics League (RCLL) was developed to provide a possibility to analyze this problem in a more simplified environment[NLF15]. In the game of the RoboCup Logistics League, two teams competing against each other utilize robots to fulfill orders under time constraints by using their environment. Each produced order is awarded with points, the team scoring the highest wins [DNK+15].

The latest developments in Satisfiability Modulo Theories (SMT) solving, which resulted in a significant improvement in runtime for commonly occuring cases [CKJ+15], led to the consideration of solving the problem of managing and optimizing the interaction between autonomous units by means of an SMT-solver [NLLÁar].

In order to examine this potential solution in more detail, this thesis presents an approach to generate an action plan for robots playing the RoboCup Logistics League game. A plain [dMB08] as well as an *optimizing* SMT-solver [BPF15] are used to meet the game's goal of scoring the highest points and are subsequently evaluated.

2 Preliminaries

This chapter gives a brief overview about the used technology of SMT-solving, followed by a description of the RoboCup Logistics League game.

2.1 SMT-Solving for Quantifier-Free Linear Integer Arithmetic

The present work makes use of an SMT-solver to find fulfilling assignments for Quantifier-Free Linear Integer Arithmetic (QF-LIA) formulas. In the following QF-LIA formulas are introduced and SMT-solving is roughly explained.

2.1.1 Quantifier-Free Linear Integer Arithmetic

The following rules in Backus–Naur form explain the construction of QF-LIA formulas:

$$\begin{aligned} f &::= c \mid \neg f \mid (f \wedge f) \mid (f \vee f) \mid (f \rightarrow f) \\ c &::= t = t \mid t > t \mid t < t \mid t \geq t \mid t \leq t \\ t &::= n \mid d \cdot x \mid t + t \end{aligned}$$

with n and d a integer constants, and x a integer-valued variable.[BFT16] The symbol f denotes formulas, c constraints, and t terms. The semantics of the used symbols are defined in the common way. It is to mention that the operators $\vee$ and $\rightarrow$ and the predicates $\geq$, $\leq$ and $<$ are syntactic sugar can be derived from the operators $\wedge$ and $\neg$ and the predicates $>$ and $=$.

The above introduced formulas with their corresponding semantics belong to the quantifier-free fragment of the first order logic and are constructed as boolean combinations of linear integer arithmetic constraints.

2.1.2 SMT-Solving

SMT-solving aims at checking whether a given formula in first order logic or its fragment is satisfiable, with respect to a given theory. That means, a certain meaning is already assigned to some functions and predicates of the formula prior to the solving process.

The most modern SMT-solvers embed a SAT-solver which searches for a solution of the boolean structure of the given formula first and secondly tries to extend the solution to satisfy the formula with respect to the given theory. A boolean skeleton is generated by replacing each constraint by a new

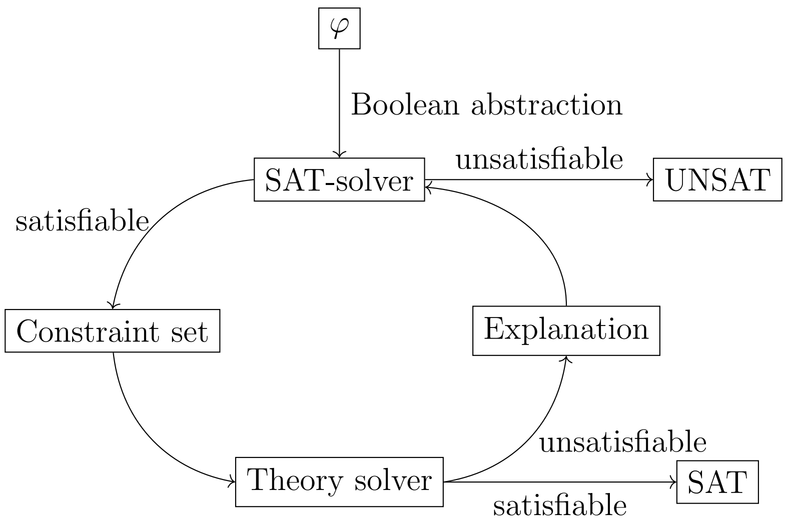


Figure 1: Lazy SMT-solving

boolean variable to achieve a solution for the boolean structure of the formula by means of the SAT-solver. The boolean skeleton's unsatisfiability leads to the formulas unsatisfiability, otherwise the set of constraints corresponding to the gained assignment is checked for consistency by a theory solver. In the case that the constraint's inconsistency is found, an explanation, usually in the form of an infeasible subset of the constraints, is given back to the SAT-solver. By means of this information another boolean solution is searched and the above described steps are performed again. If all combinations of constraints, which satisfy the boolean skeleton are not consistent the formula is not satisfiable regarding the given theory. The presented approach is called *lazy* SMT-solving, a *less lazy* approach uses only partial solutions of the boolean structure to make use of the faster "learning" smaller incremental steps provide.

There are SMT-solvers available which extend their functionality to optimize solutions in regards to some criteria for QF-LIA formulas. The optimizer is able to search for a solution maximizing or minimizing given terms.

The above mentioned information originates from [ÁLCS10].

2.2 The RoboCup Logistics League

The aim of the RoboCup Logistics League game is to simulate problems which are representative for the problems occurring when shifting from static process chains to smart factories. These include the problem of coordinating the interactions of the autonomous processing units to enable and optimize the production. Since the present thesis is meant to address this challenge, in the following the description of the game is restricted to the important parts needed to solve this task.

Two teams compete against each other on a defined playing field. Autonomous robots are used to transport material between machines performing processing steps to create products. In the course of the competition orders are announced, demanding certain products. The delivery of these goods as well as their production is awarded with points unless the corresponding order's deadline is violated. The team scoring the highest wins.

A product is composed of a *base* element, zero to three intermediate *rings*, and a *cap* as the topmost component. A particular component is of a specific color. Stations are placed on the playground enabling the dispensing of bases, the mounting of rings or caps, and the delivery of the finished product. The robots are used to take one intermediate product at a time to the stations and initiate a particular processing step on it. Each team owns three robots, starting from a designated area, one base station, two ring stations, two cap stations and one delivery station, randomly placed on the playground.

Base Station (BS) dispenses base elements colored in red, black, or silver.
Ring Station (RS) mounts a ring on a intermediate product, one specific Ring Station is able to mount two distinct colors. Some colors need up to two additional bases fed into the machine prior to the mounting process. Before feeding the additional bases or starting the mounting process the station has to be set up for the color to be mounted. Available ring colors are blue, green, yellow, and orange.
Cap Station (CS) mounts a cap on an intermediate product. As the first step, a robot needs to feed a cap mounted on a transparent base, originating from the station's shelf, into the cap station. After the station dismounted the cap and the transparent base located in the station's output is removed, the cap is ready to mount. The transparent base may be dropped or used to be fed as an additional base to a ring station; products containing transparent bases are not possible. A cap station has either black or gray colored caps located on its shelf.
Delivery Station (DS) accepts completed products.

Type	Distribution	(Final) processing time[s]
Base Station (BS)	1 per team	minimum physical time
Cap Station (CS)	2 per team	15 to 25 sec
Ring Station (RS)	2 per team	40 to 60 sec
Delivery Station (DS)	1 per team	20 to 40 sec

Table 1: Processing times, originating from [DNK+15]

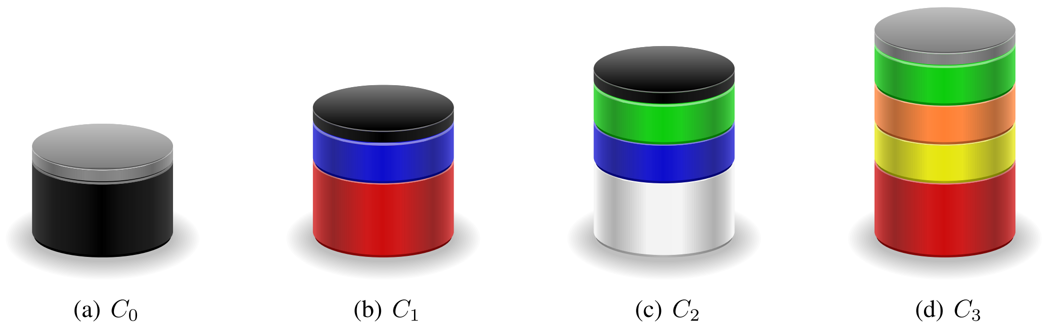
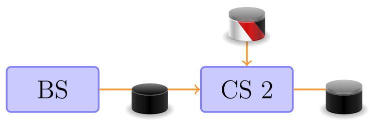

(a) C_0 (b) C_1 (c) C_2 (d) C_3

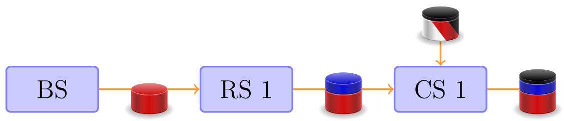
Figure 2: Product complexity levels, originating from [DNK+15]

Depending on the amount of rings mounted on a product, complexity levels are defined. A product consisting of a base and cap belongs to the complexity level C_0, if one ring is mounted it belongs to level C_1; the complexity levels C_2 and C_3 are defined analogously. For the ring colors, additional complexity levels are defined: CC_0, CC_1, CC_2, denoting whether zero, one or two additional bases are needed.

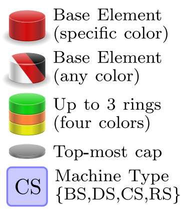
The steps of feeding an additional base into a ring station, the mounting of a ring or cap, and the delivery of a product is rewarded. The higher the complexity level of a product is, the higher the points awarded for performing the corresponding intermediate steps. The points for performing intermediate steps are only awarded if the product is delivered. [DNK+15] There are several other steps rewarded during the game, for simplification they are not considered in this thesis.



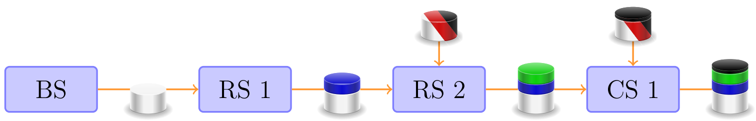
(a) Production of complexity C_0 only mounting a cap on a BE as shown in Figure 10(a).



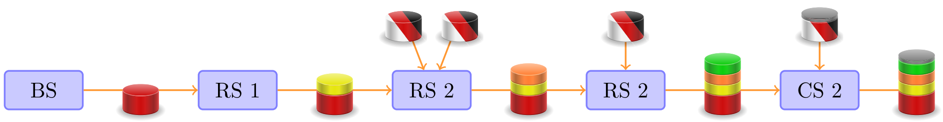
(b) Production of complexity C_1 with a single intermediate ring according to Figure 10(b).


Base Element (specific color)
Base Element (any color)
Up to 3 rings (four colors)
Top-most cap
CS
Machine Type {BS,DS,CS,RS}

(c) Legend



(d) Complexity C_2 with two rings (cf. Figure 10(c)). RS 2 requires an additional base (of any color) for the green ring.



(e) Production of the highest complexity C_3 with a three intermediate rings according to Figure 10(d). Note that RS 2 requires two additional BE (of any color) for the orange ring and another one for the green ring.

Figure 3: Progression steps, originating from [DNK+15]

Sub-task	Production Phase	Points
Additional base	Feed an additional base into a ring station	+2
Finish CC_0 step	Finish the work order for a color requiring no additional base	+5
Finish CC_1 step	Finish the work order for a color requiring one additional base	+10
Finish CC_2 step	Finish the work order for a color requiring two additional bases	+20
Finish C_1 pre-cap	Mount the last ring of a C_1 product	+10
Finish C_2 pre-cap	Mount the last ring of a C_2 product	+30
Finish C_3 pre-cap	Mount the last ring of a C_3 product	+80
Mount cap	Mount the cap on a product	+10
Delivery	Deliver one of the final product variants to the designated loading zone at the time specified in the order	+20

Table 2: Reward, originating from [DNK+15]

3 General Approach

As described in the introduction the aim is to utilize an optimizing SMT-solver to generate an action plan for robots playing the RoboCup Logistic League game. Therefore, in the following section the concrete RoboCup Logistics League planning problem will be abstracted in order to allow the easier derivation of a procedure that generates an SMT-formula whose optimal model can be interpreted as the optimal action plan for a given initial situation.

3.1 Problem Abstraction

Let an environment, the current state of the environment, a set of possible actions, and a rating of the different environment states be given. A valid action sequence of a fixed length leading to the highest rated world state is searched. One way of solving this scheduling problem by means of an optimizing SMT-solver is by creating a formula φ with the following properties:

- the existence of a bijective mapping from models of φ to the action sequences that are possible on the basis of the current world state,
- a term in φ expresses the rating of the final world state resulting from the action sequence that is represented by a model of the formula.

Since the mapping pairs each possible action sequence with a model of the formula, finding a model which maximizes the value of the rating-term is equivalent to finding an action sequence leading to the highest rated world state. The above described properties are visualized in Figure 4.

3.2 Solving Approach

In the following section, a possibility to regard special sequences of world states as action sequences is introduced. This enables the interpretation of formula models as action sequences by using the intermediate step of interpreting the models as sequences of world states before. Subsequently the creation of a formula and a corresponding interpretation of the formula's models as world state sequences will be described.

Interpreting Valid World State Sequences as Action Sequences It is assumed that actions are unambiguously characterized by their caused alternation of the environment and the preconditions allowing their execution.

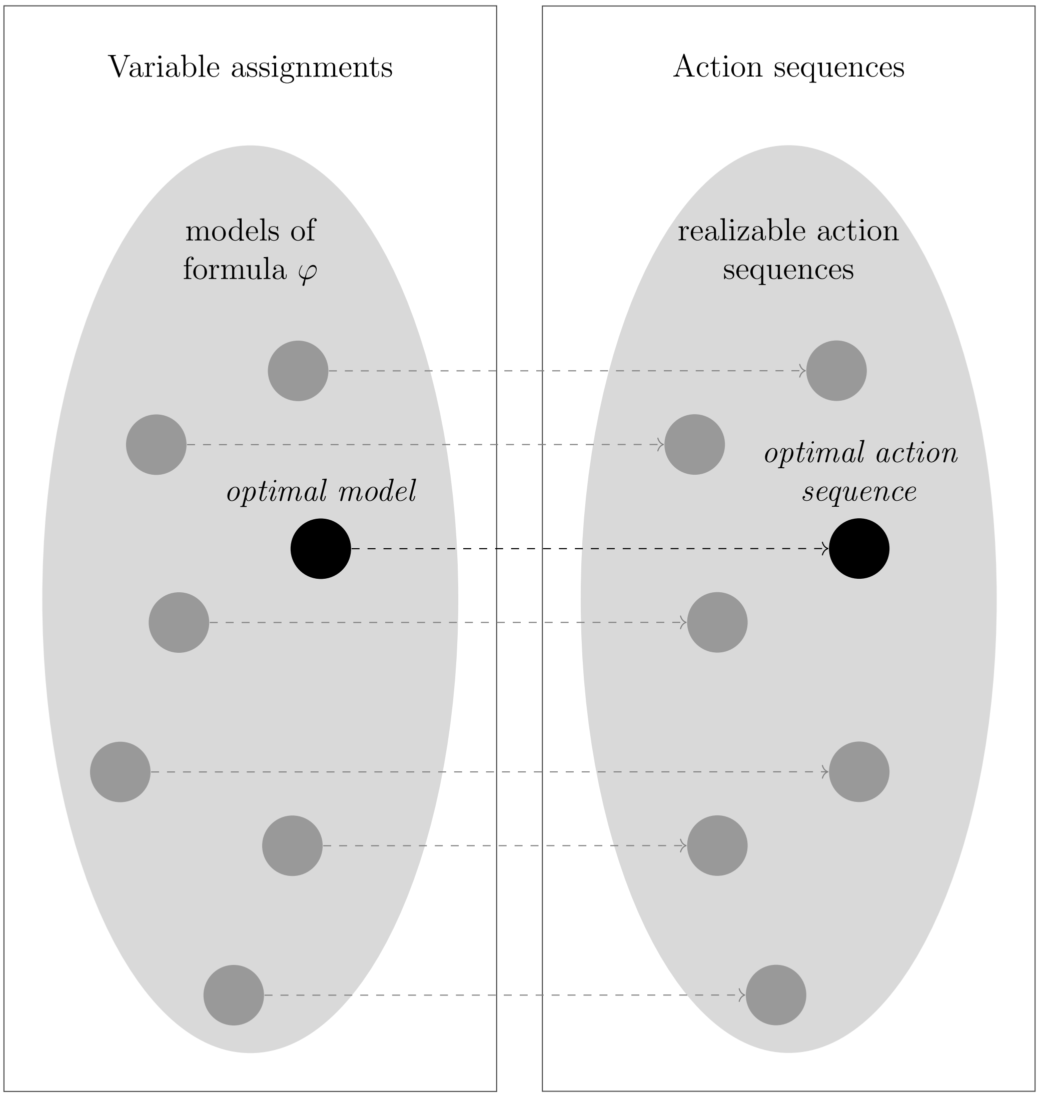

Variable assignments
Action sequences
models of formula φ
realizable action sequences
optimal model
optimal action sequence

Figure 4: Properties of φ

Given a set of possible actions, a sequence of world states can be interpreted as a realizable sequence of actions if the following properties are fulfilled:

- A world state is followed by a second world state only if the latter is induced by an action on the previous world state,
- The previous world state fulfills all preconditions of the action.

Two consecutive world states will be interpreted as a particular action if the first world state fulfills the action's preconditions and the following world state is induced by the action. In the following, action sequences fulfilling the above properties will be referred to as *valid* world state sequences.

As a result the task "finding a bijective mapping from the formula models to *possible action sequences*" can be reformulated to "finding a bijective mapping from the formula models to *sequences of world states fulfilling the above mentioned properties*".

World State Representation To achieve the above mentioned objective, the creation of variables and the interpretation of their assignments to world state sequences is described. For each possible world state in such a sequence a set of variables will be created. A variable contained in a set indicates a specific important aspect of the environment. Therefore a variable has a counterpart in all other sets, which represent the same aspect of the environment but in another step in the planning. A particular value assigned to a variable describes a particular state of the environment aspect which is denoted by the variable.

Interpreting Models of φ as Valid World State Sequences The following paragraph describes the construction of the formula φ under the objective that models of φ represent a sequence of world states which can be interpreted as a realizable action sequence as described before. Furthermore, on the one hand it has to be ensured that only world state sequences which can be considered as realizable action sequences follow from models of the φ, on the other hand for each realizable action sequence a corresponding model of φ has to exist.

In the following it is assumed that the searched action sequence is of the length $k+1 \in \mathbb{N}$ and A represents the set of actions which can be performed in the environment. For each possible action $a \in A$ and each position $i \in \{0, .., k\}$ in the action sequence a formula φ_a^i is created. Thereby i also denotes the position of the world state the action is performed on the underlying world state sequence and $i+1$ the position of the resulting world state.

A model of φ fulfilling φ_a^i forces the placement of action a on position i in the interpreted action sequence. In order to enable the above mentioned properties, models of φ_a^i have to ensure that the preconditions for a holds in the world state representing the starting situation of a. Additionally it has to be guaranteed that the following world state represents the environment alternation induced by a on the previous world state, while the unchanged aspects are inherited from the previous world state.

It has to be mentioned that a model cannot fulfill more than one of the action-formulas φ_a^i for a particular position, since an action is unambiguously defined by its preconditions and resulting environment alteration.

It follows that a model fulfilling the disjunction of the formulas φ_a^i for a fixed position i

$$\varphi^i := \bigvee_{a \in A} \varphi_a^i$$

forces that *exactly one* action from the set of possible actions A is placed on position i in the interpreted action sequence, while it is ensured that the action on position $i-1$ results in a world state fulfilling the preconditions of the action on position i.

In order to force all positions $i \in \{0, ..., k\}$ of the interpreted action sequence to be determined unambiguously the model has to fulfill all formulas φ^i. Therefore the model has to fulfill the conjunctions over the formulas, which each represents an action on a particular position:

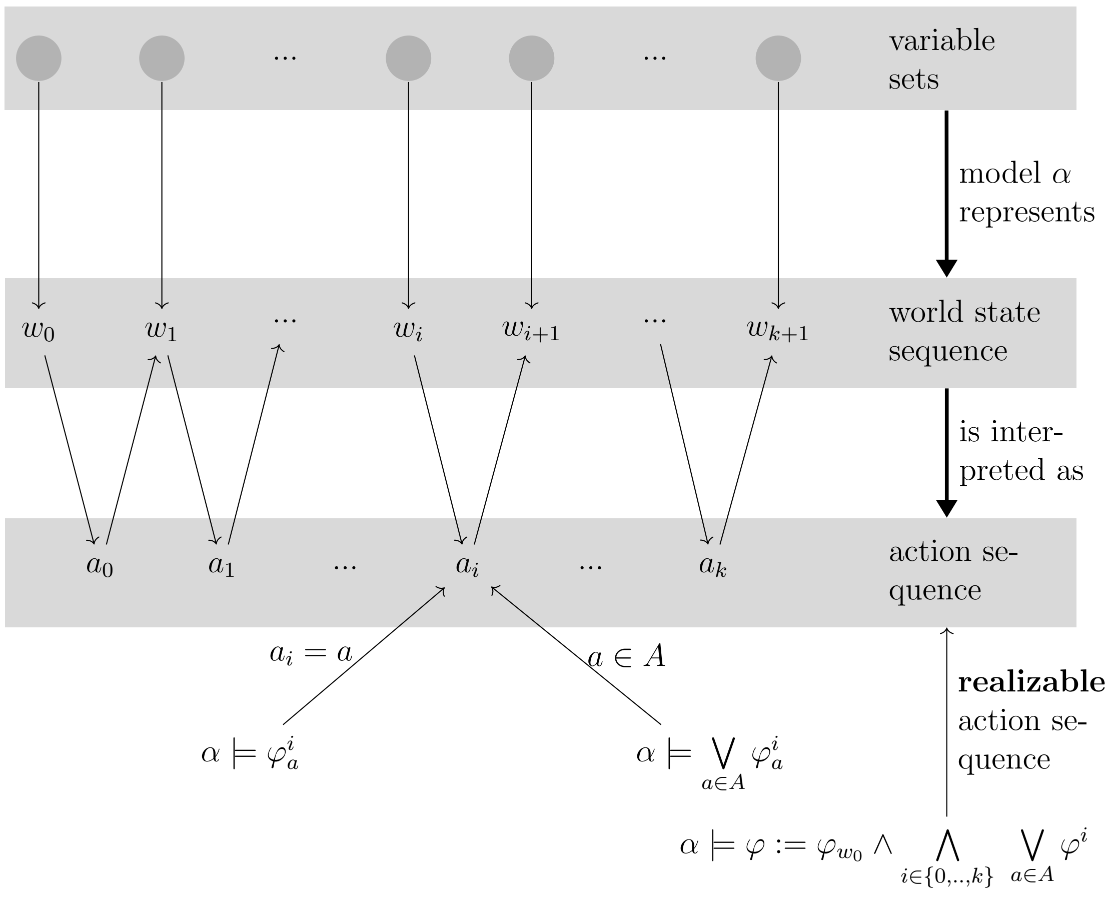
$$\bigwedge_{i \in \{0,..,k\}} \varphi^i = \bigwedge_{i \in \{0,..,k\}} \bigvee_{a \in A} \varphi_a^i$$

Since the first action in all possible action sequences has to fulfill the initial world state w_0, an additional formula φ_{w_0} has to be created which ensures that the variable set representing the first position in the corresponding world state sequence is assigned with values representing w_0.

This leads to the formula φ, whose models represent all possible action sequences of length k over the possible actions A beginning with the initial world state w_0:

$$\varphi := \varphi_{w_0} \wedge \bigwedge_{i \in \{0,..,k\}} \bigvee_{a \in A} \varphi_a^i$$

The procedure of creating φ and the corresponding interpretation steps are visualized in Figure 5.



variable sets
model α represents
w_0 w_1 ... w_i w_{i+1} ... w_{k+1}
world state sequence
is interpreted as
a_0 a_1 ... a_i ... a_k
action sequence
$a_i = a$
$a \in A$
$\alpha \models \varphi_a^i$
$\alpha \models \bigvee_{a \in A} \varphi_a^i$
realizable action sequence
$\alpha \models \varphi := \varphi_{w_0} \wedge \bigwedge_{i \in \{0,..,k\}} \bigvee_{a \in A} \varphi^i$

Figure 5: Construction and interpretation of φ

4 RoboCup Logistics League Specific Encoding

In the previous chapter formula properties were derived to allow the solving of scheduling problems similar to the RoboCup Logistics League planning problem by means of an optimizing SMT-solver in *general*. The present chapter aims at introducing a procedure to generate *concrete* formulas for the RoboCup Logistic League planning problem which have the characteristics described in the chapter before. More precisely, first, the aspects of the RoboCup Logistics League environment, which are important to consider for the planning, are identified and variables are assigned to each dynamic aspect. Secondly, a set of actions is abstracted from the actual interaction possibilities of the robots with the environment, since considering actions on the level of controlling the different modules of a robot would not be beneficial. Subsequently, formulas are described representing the previously introduced actions in a step of the planning. Finally, the advantages and disadvantages of several reward representations are discussed and documented with measurement results of their solving time.

4.1 World State Encoding

This section starts by distinguishing between environment aspects which are fixed before the beginning of the game and hence influence the creation of the formula and those aspects which may change during the game and thus have to be represented by variables. Further, the encoding of the time aspects of an arbitrary action-formula are described and the corresponding variables are introduced. The description of the representation of the existence or coloring of workpieces follows. The variables encoding the remaining world state aspects are presented in the section describing the action encoding.

Dynamic Aspects The following important dynamic aspects have to be represented by variables:

- the point in time until a particular machine is blocked due to the execution of a refinement step or a robot operating this machine,
- the time a robot needs to reach a particular machine,
- whether the cap station is fed with a cap,
- the color of the fed cap, if the cap station is fed with a cap,

- whether the ring station is configured for mounting a ring
- the color of the ring to be mounted if the ring station is configured for mounting a ring,
- the additional bases a ring station needs,
- the points scored.

Static Aspects Static aspects which have to be considered while creating the action formulas are:

- the processing time for a refinement step,
- the time a robot needs to move from one machine to another,
- the amount of additional bases a ring station needs for mounting a ring in a particular color,
- the ordered products,
- the deadline of an order,
- reward points gained for the various refinement steps and the delivering of ordered products with various complexity,
- the colors of the components a station is able to mount or to deliver.

The arrival of an order and the product demanded is not known in advance, therefore not yet arrived orders can not be considered in the calculation of a scheduling. Therefore, the information about the ordered products is static in terms of the scheduling.

Simplifications Several simplifications about the game are made. It is assumed that the differences to reality caused by this simplification have little impact on the optimality of the gained action plan:

- The time needed to move between machines is assumed to be constant, therefore the time required for collision prevention between robots is not considered.
- If a machine is broken and the time when the machine is properly functioning again is known in advance, encode that the machine is blocked until it is repaired in the initial world state, otherwise act as if the machine is non-existent for the scheduling.

Notations In the following, let R denote the set of robots, S_{BS} the set of base stations, S_{RS} the set of ring stations, S_{CS} the set of cap stations, S_{DS} the set of delivery stations, and O the set of currently existing orders. Whereas $S = S_{BS} \cup S_{RS} \cup S_{CS} \cup S_{DS}$ denotes the set of all stations and $M = R \cup S$ denotes the set of all machines.

In all following formula descriptions $k + 1 \in \mathbb{N}$ denotes the length of the searched action sequence and variables of the form $X^i_{\_}$ represent an environment property of the world state on position $i \in \{0, ..., k + 1\}$ in the world state sequence.

4.1.1 Time Aspects

Since there exists only one station for each component color, the interaction times of two robots on a station may interfere. A robot may only interact with a station at a time when the station is not blocked and when the robot is not occupied. Thereby one has to consider the travel time of the robot to the particular station, the time the robot needs to handle the workpiece, and the processing time of the production step.

Station Occupation If a base is collected from a base station, it is blocked for the time it takes to dispense the base and the time the robot needs to collect the base. The time required to feed a cap from the shelf to the cap station consists of the time needed to take the cap and deposit it on the input conveyor belt. Dropping a transparent base from the output of a cap station blocks the station approximately for the time it takes to grab the base. Assuming that a robot holds a base, a ring station is blocked until deposition of the base on the conveyor belt is finished in case that the action of feeding an additional base is performed. A station is blocked the time it takes for the robot to deposit the workpiece on the machine and the time the production steps needs to finish if the action of mounting a ring or cap is performed. Collecting a workpiece from the output of a station blocks the station for the time the robot needs to pick up the workpiece. The delivering station is blocked for the time it takes to deposit the product.

The fact that a robot cannot operate a station if another robot has performed the actions of handling the workpiece but has not moved yet is ignored. It is assumed that the time of driving away is so low that ignoring it has no impact to the scheduling. Further it is assumed that a robot frees the space in front of a machine if the robot is idle, in this case the potential differences in travel times caused by the new placement of the robot are not considered.

Time Encoding In the following, the creation of a formula $\varphi^i_{time(a,r,s,d)}$ is introduced, which represents the influence an action a has on the occupation time of the involved station $s \in S$ and the travel times of the involved robot $r \in R$ in the ith step of the action plan. The differentiation whether the station is approached from the input or output side is represented by $d \in \{in, out\}$.

To represent the occupation of a station $s \in S$ the variable $X^i_{occ(s)}$ is introduced. Its value indicates the point in time the occupation of the station s ends. Likewise the variable $X^i_{mov(r,s,d)}$ with $s \in S$ and $r \in R$ encodes the earliest possible arrival of the robot r on the station s on the side $d \in \{in, out\}$. Additionally, let $t_{mov(s,d,s',d')}$ denote the time a robot needs to move from the side d of station s to the side d' of station s' with $s' \in S$; $t_{process(a)}$ denotes the processing time the action induces on on the machine involved in a; and $t_{handling(a)}$ denotes the time robot r needs to handle the workpiece when performing the action.

A robot is at earliest able to operate a station by arrival at such. In case the station is occupied at this time the robot has to wait till the station is idle again. The situation of the robot arriving at an idle machine is expressed by the constraint:

$$X^i_{occ(s)} \leq X^i_{mov(r,s,d)}$$

In the resulting world state, the station is idle again after the robot arrived, handled the workpiece and, if a production step is performed, the expiry of the processing time:

$$X^{i+1}_{occ(s)} = X^i_{mov(r,s,d)} + t_{handling(a)} + t_{process(a)}$$

Since a specific action's precondition and result has to hold if it is performed then the conjunction of the above mentioned constraints has to hold:

$$X^i_{occ(s)} \leq X^i_{mov(r,s,d)} \wedge X^{i+1}_{occ(s)} = X^i_{mov(r,s,d)} + t_{handling(a)} + t_{process(a)}$$

Additionally, the time the robot is able to arrive at a specific side of a station in the following world state is now the addition of the time the robot needs to reach its current destination and the travel time between these and the particular station:

$$\bigwedge_{s' \in S} \bigwedge_{d' \in \{in,out\}} X^{i+1}_{mov(r,s',d')} = X^i_{mov(r,s,d)} + t_{mov(s,d,s',d')}$$

In the case that the robot reaches a station while it is occupied and has to wait, expressed by the constraint

$$X^i_{occ(s)} > X^i_{mov(r,s,d)}$$

the occupation time in the next step is determined by the end of the current occupation, the work piece handling time and the processing time:

$$X^{i+1}_{occ(s)} = X^{i}_{occ(s)} + t_{handling(a)} + t_{process(a)}$$

Therefore, to encode the circumstance that the robot has to wait until an occupied station is free again both constraints have to be satisfied:

$$X^{i}_{occ(s)} > X^{i}_{mov(r,s,d)} \wedge X^{i+1}_{occ(s)} = X^{i}_{occ(s)} + t_{handling(a)} + t_{process(a)}$$

Should this case arise, the time the engaged robot is able to reach a particular station side depends on the time the station is occupied, respectively the robot has to wait and the time the robot requires to handle the workpiece and move from the used station's side to the next station's side:

$$\bigwedge_{s' \in S} \bigwedge_{d' \in \{in,out\}} X^{i+1}_{mov(r,s',d')} = X^{i}_{occ(s)} + t_{handling(a)} + t_{mov(s,d,s',d')}$$

The above mentioned coherence is described by the disjunction of the conjunction of the constraints belonging to one case:

$$\begin{aligned} &\varphi^{i}_{time(a,r,s,d)} := \\ &(X^{i}_{occ(s)} \leq X^{i}_{mov(r,s,d)} \wedge X^{i+1}_{occ(s)} = X^{i}_{mov(r,s,d)} + t_{handling(a)} + t_{process(a)} \\ &\quad \wedge \bigwedge_{s' \in S} \bigwedge_{d' \in \{in,out\}} X^{i+1}_{mov(r,s',d')} = X^{i}_{mov(r,s,d)} + t_{handling(a)} + t_{mov(s,d,s',d')}) \\ &\vee (X^{i}_{occ(s)} > X^{i}_{mov(r,s,d)} \wedge X^{i+1}_{occ(s)} = X^{i}_{occ(s)} + t_{handling(a)} + t_{process(a)} \\ &\quad \wedge \bigwedge_{s' \in S} \bigwedge_{d' \in \{in,out\}} X^{i+1}_{mov(r,s',d')} = X^{i}_{occ(s)} + t_{handling(a)} + t_{mov(s,d,s',d')}) \end{aligned}$$

If the robot arrives at an idle station the first part of the disjunction holds, otherwise the second part is satisfied. In both cases the time-variables of the following world state are forced to be assigned with the correct values. This circumstance is pictured in Figure 6.

Alternative Encoding The way of expressing the restriction of the robots to operate a station by means of time aspects can be further abstracted. That is, instead of differentiating between *the time it takes for a robot to reach a particular station* and *a given station's occupied time*, the restriction can be simplified by considering only one point in time. More specifically, *the point in time at which a given robot is able to operate a particular machine*, is sufficient to express all required time aspects without the loss of important information for the scheduling.

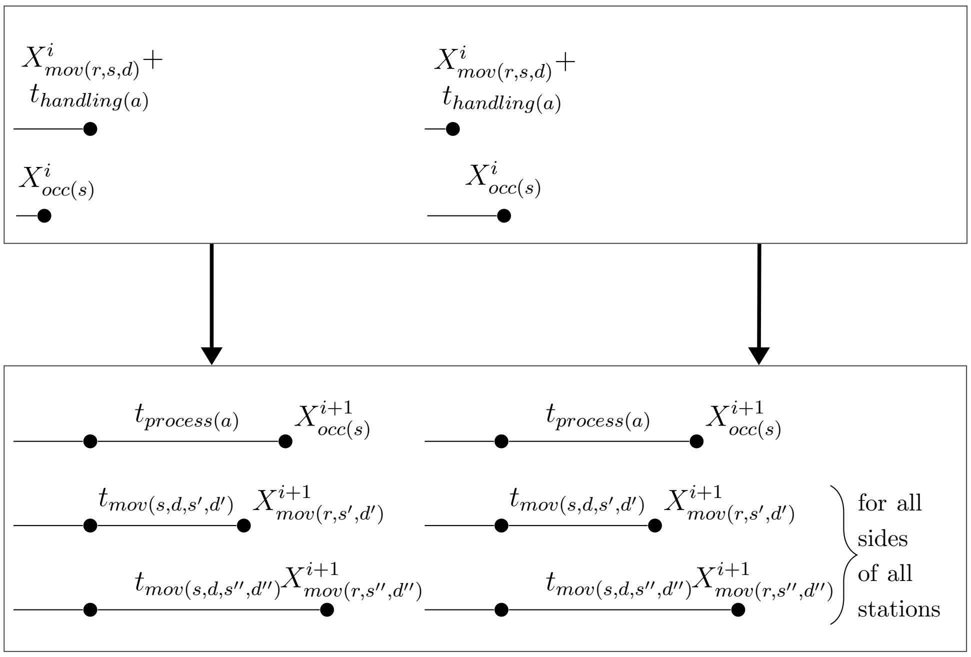


Figure 6: Time encoding

In the following, let $r \in R$ be the robot involved in the action, $s \in S$ be the operated station, and $d \in \{in, out\}$ be the used side of the station s.

The variable $X^i_{op(r,s,d)}$ is introduced whose value represents the point in time at which the robot r is available for operating the station s on its side d.

Given a world state and an action performed on it, the involved robot r is soonest able to operate a station $s' \in S$ on side $d' \in \{in, out\}$ in the following world state after handling the workpiece for the current action and moving to s':

$$X^{i+1}_{op(r,s',d')} = X^i_{op(r,s,d)} + t_{handling(a)} + t_{mov(s,d,s',d')}$$

The possibility that s' was previously involved in another action and is occupied for a longer time than the time r needs to arrive at the side d' of s' has to be considered:

$$X^i_{op(r,s',d')} \geq X^i_{op(r,s,d)} + t_{handling(a)} + t_{mov(s,d,s',d')} \land \; X^{i+1}_{op(r,s',d')} = X^i_{op(r,s',d')}$$

This leads to the following formula considering all sides of all stations:

$$\bigwedge_{s' \in S} \bigwedge_{d' \in \{in,out\}} ((X^i_{op(r,s',d')} < X^i_{op(r,s,d)} + t_{handling(a)} + t_{mov(s,d,s',d')}$$
$$\wedge\ X^{i+1}_{op(r,s',d')} = X^i_{op(r,s,d)} + t_{handling(a)} + t_{mov(s,d,s',d')})$$
$$\vee\ (X^i_{op(r,s',d')} \geq X^i_{op(r,s,d)} + t_{handling(a)} + t_{mov(s,d,s',d')}$$
$$\wedge\ X^{i+1}_{op(r,s',d')} = X^i_{op(r,s',d')}))$$

An action during which the robot r occupies the station s influences the point in time another robot $r' \in R$ is able to operate s in the next step if r''s current operating time for s is before the point in time to which r occupies s:

$$X^i_{op(r',s,d')} < X^i_{op(r,s,d)} + t_{handling} + t_{process(a)}$$
$$\wedge\ X^{i+1}_{op(r',s,d')} = X^i_{op(r',s')} + t_{handling(a)} + t_{process(a)}$$

Otherwise r's occupation time for s' does not change from the current to the following step:

$$X^i_{op(r',s,d')} \geq X^i_{op(r,s,d)} + t_{handling(a)} + t_{process(a)}$$
$$\wedge\ X^{i+1}_{op(r',s,d')} = X^i_{op(r',s,d')}$$

From the above the following formula results which encodes the time constraints of the game for an action:

$$\varphi^i_{time(a,r,s,d)} :=$$
$$\bigwedge_{s' \in S} \bigwedge_{d' \in \{in,out\}} ((X^i_{op(r,s',d')} < X^i_{op(r,s,d)} + t_{handling(a)} + t_{mov(s,d,s',d')}$$
$$\wedge\ X^{i+1}_{op(r,s',d')} = X^i_{op(r,s,d)} + t_{handling(a)} + t_{mov(s,d,s',d')})$$
$$\vee\ (X^i_{op(r,s',d')} \geq X^i_{op(r,s,d)} + t_{handling(a)} + t_{mov(s,d,s',d')}$$
$$\wedge\ X^{i+1}_{op(r,s',d')} = X^i_{op(r,s',d')}))$$
$$\wedge \bigwedge_{r' \in R \setminus \{r\}} \bigwedge_{d' \in \{in,out\}} ((X^i_{op(r',s,d')} < X^i_{op(r,s,d)} + t_{handling(a)} + t_{process(a)}$$
$$\wedge\ X^{i+1}_{op(r',s,d')} = X^i_{op(r',s')} + t_{handling(a)} + t_{process(a)})$$
$$\vee\ (X^i_{op(r',s,d')} \geq X^i_{op(r,s,d)} + t_{handling(a)} + t_{process(a)}$$
$$\wedge\ X^{i+1}_{op(r',s,d')} = X^i_{op(r',s,d')}))$$

Comparison To compare the quality of both encodings one has to consider that the increased amount of disjunctions in the second encoding has a negative effect on the solving time an SMT-optimizer needs, since this results in a greater amount of models for the boolean skeleton of the formula. On the other hand the reduced number of variables may have a positive influence on the solving times. In conclusion to rate the quality of the encodings one has to perform practical tests.

It has to be mentioned that both encodings consider the aspects of time in the game as points in time rather than time spans. That means an action which occupies a station only for a small time span will mark the station from the start of the scheduling till the end of the action as occupied. In the following steps possible idle time slots which lay before this mark cannot be involved in the further planning. However, this circumstance does not restrict the set of possible plans delivered by the encoding. By rearranging the order of the actions such that an unused time slot is utilizable again and if this order results in a possible action sequence which leads to the highest score the corresponding model of φ will be found.

4.1.2 Workpiece Encoding

There exist several possibilities to describe the presence and composition of a workpiece in the output of a station, respectively the gripper of a robot. Only the ring and cap station's output have to be considered since the delivery station is not provided with an output belt and the action of requesting and collecting a base will be encoded as a monolithic action.

The main characteristic of the approach that has been chosen for the following encoding is that, for each of the above described machines, additional variables are introduced which specify the single components of the workpiece for each step i:

- The value of the variable B^i_m with $m \in M \setminus (S_{BS} \cup S_{DS})$ denotes whether a base is present on the machine m in step i and if this is the case, the coloring of the base.
- The value of the variable $R^i_{j,m}$ with $m \in M \setminus (S_{BS} \cup S_{DS})$ and $j \in \{1, 2, 3\}$ denotes whether the jth ring is present on the machine m in world state i and if this is the case, the coloring of the ring.
- The value of the variable C^i_m with $m \in M \setminus (S_{BS} \cup S_{DS})$ denotes whether a cap is present on the machine m in step i and if this is the case, the coloring of the cap.

The existence of an arbitrary component and its coloring is encoded by the set of integer ids of possible component colors $C \subset \mathbb{N}$. Since needed to be queried explicitly for the construction of the formulas the notation $c_{transparent} \in C$ and $c_{none} \in C$ is used to determine whether a component is transparent or nonexistent.

The consistency of the workpiece representation is ensured in the encoding of the initial state and the action formulas. That means, it has to be ensured that, if the first ring is mounted the machine has to hold a base, if the second or third ring is mounted the previous ring has to be mounted, and if the cap is mounted the machine has to hold at least a base. Therefore for all machines $m \in M$ and all world states on position i the following statements are invariant:

- $\neg(R^i_{1,m} = c_{none}) \rightarrow \neg(B^i_m = c_{none})$
- $\neg(R^i_{2,m} = c_{none}) \rightarrow \neg(R^i_{1,m} = c_{none})$
- $\neg(R^i_{3,m} = c_{none}) \rightarrow \neg(R^i_{2,m} = c_{none})$
- $\neg(C^i_m = c_{none}) \rightarrow \neg(B^i_m = c_{none})$
- $B^i_m = c_{none} \rightarrow R^i_{1,m} = c_{none} \wedge R^i_{2,m} = c_{none} \wedge R^i_{3,m} = c_{none} \wedge C^i_m = c_{none}$

The last statement allows to verify that the machine holds an arbitrary workpiece by checking for the existence of the base only.

4.1.3 Initial State

The scheduling starts from a concrete world state w_0. It has to be ensured that the first action in the action sequence can be performed on w_0, that means w_0 fulfills the precondition for it. Therefore w_0 has to be represented as the first world state in the sequence of world states following from a concrete model of φ. This is achieved by constructing constraints which force a model of φ to assign the values representing w_0 to the corresponding variables of the variable set which represents the first world state. In the following, the conjunction over these constraints is denoted by φ_{w_0}.

4.2 Action Encoding

The actual interaction possibilities of a robot with the environment are very fine grained. It is not necessary to consider these interactions as single tasks in the scheduling to achieve optimal results. Instead, it is sufficient to group the interactions of moving and commanding the grippers of a robot as distinct actions. Otherwise, considering all interaction possibilities as distinct actions in our scheduling would require the representation of all resulting world states as variable sets. This would lead to a large amount of variables and a huge φ, influencing the solving time negatively. In the following, only sequences of interactions of a particular robot leading to an alteration of a station status will be considered as actions. Therefore, the moving to a station and the controlling of the robots grippers are implicitly contained in these actions.

Actions

- collecting a base of a particular coloring from a base station,
- mounting a cap
  - feeding a cap from the shelf into a cap station,
  - mounting a cap by feeding a workpiece into a cap station,
- mounting a ring
  - specify the color of the ring a ring station will mount on the next fed workpiece,
  - feeding a base to a ring station if required for a particular ring color,
  - mounting a ring by feeding a workpiece to a ring station,
- collecting a workpiece from the output of a station after the station processed the workpiece,
- dropping a transparent base from the output of a cap station,
- delivering a completed product a robot holds.

The dropping of an arbitrary workpiece a robot holds is not considered for now. The including of this action would greatly increase the amount of possible action sequences, likely leading to a much higher solving time.

In the following let A denote the set of the above derived actions.

Unaffected Properties In the following description of the action-formulas only constraints are listed which determine changing aspects of the resulting world state. If an aspect is not affected by an action, the corresponding constraint will not be mentioned explicitly. Instead, all constraints of the form $X_{\_}^{i+1} = X_{\_}^{i}$ with $X_{\_}^{i}$ denoting a variable representing an unchanging aspect of the world state the action is performed in and $X_{\_}^{i+1}$ representing the same aspect in the following world state will be combined in formulas $\varphi_{rem^i(a,r,s)}$, $a \in A$ denotes the performed action, $r \in R$ the involved robot, $s \in S$ the used station, and $i \in \mathbb{N}$ the step of the action plan.

Reward Representation Possibilities to represent the points gained during the game for producing and delivering orders are discussed after the action-formulas are presented. The formula $\varphi^i_{order(a,r,s)}$ denotes this reward representation for each action $a \in A$, robot $r \in R$ and station $s \in S$ in the action-formula presentations in the ith step of the action plan.

Action-Formulas Let C_s denote the set of component colors the station $s \in S_{BS} \cup S_{RS} \cup S_{CS}$ is able to mount, respectively deliver in case of a base station and let $c_{base} : O \mapsto C$, $c_{ring,i} : O \mapsto C$, and $c_{cap} : O \mapsto C$ denote the base coloring, the coloring of the ith ring, and the cap coloring of the product requested by an order.

Each of the following subsections present a formula for one of the above mentioned actions performed by robot $r \in R$. Since collecting a base, feeding a cap, and setting up a ring color requires the information about the coloring $c \in C$ of the component to be mounted, the corresponding formulas are dependent on c. The formulas are denoted by the following expressions with $bs \in S_{BS}$, $rs \in S_{RS}$, $cs \in S_{CS}$, $ds \in S_{DS}$:
$\varphi^i_{collectBase(r,bs,c)}$, $\varphi^i_{setupRingColor(rs,c)}$, $\varphi^i_{feedCap(r,cs,c)}$, $\varphi^i_{mountCap(r,cs)}$, $\varphi^i_{dropTransparentBase(r,cs)}$, $\varphi^i_{feedBase(r,rs)}$, $\varphi^i_{mountRing(r,rs)}$, $\varphi^i_{collectWorkpiece(r,s)}$, $\varphi^i_{deliverWorkpiece(r,ds)}$

Since the possibility exists that any robot can perform these actions on any corresponding station, the action-formulas for each of those combinations have to be considered. Additionally, if an action is dependent on the information about the involved component's coloring, all possible colors have to be considered, unless it is foreseeable that the action is never required in order to serve an order. This leads to the following formulas:

$$\varphi^i_{collectBase} := \bigvee_{r\in R} \bigvee_{bs\in S_{BS}} \bigvee_{\substack{c\in C_{bs},\\ \exists o\in O\\ c_{base}(o)=c}} \varphi^i_{collectBase(r,bs,c)},$$

$$\varphi^i_{setupRingColor} := \bigvee_{r\in R} \bigvee_{rs\in S_{RS}} \bigvee_{\substack{c\in C_{rs},\\ \exists o\in O\\ \exists i\in\{1,2,3\}\\ c_{ring,i}(o)=c}} \varphi^i_{setupRingColor(r,rs,c)},$$

$$\varphi^i_{feedCap} := \bigvee_{r\in R} \bigvee_{cs\in S_{CS}} \bigvee_{\substack{c\in C_{cs},\\ \exists o\in O\\ c_{cap}(o)=c}} \varphi^i_{feedCap(r,cs,c)},$$

$$\varphi^i_{mountCap} := \bigvee_{r\in R} \bigvee_{cs\in S_{CS}} \varphi^i_{mountCap(r,cs)},$$

$$\varphi^i_{dropTransparentBase} := \bigvee_{r\in R} \bigvee_{cs\in S_{CS}} \varphi^i_{dropTransparentBase(r,cs)},$$

$$\varphi^i_{feedBase} := \bigvee_{r\in R} \bigvee_{rs\in S_{RS}} \varphi^i_{feedBase(r,rs)},$$

$$\varphi^i_{mountRing} := \bigvee_{r\in R} \bigvee_{rs\in S_{RS}} \varphi^i_{mountRing(r,rs)},$$

$$\varphi^i_{collectWorkpiece} := \bigvee_{r\in R} \bigvee_{s\in S_{CS}\cup S_{RS}} \varphi^i_{collectWorkpiece(r,s)},$$

$$\varphi^i_{deliverWorkpiece} := \bigvee_{r\in R} \bigvee_{ds\in S_{DS}} \varphi^i_{deliverWorkpiece(r,ds)}$$

As a result the searched formula φ can be defined as

$$\varphi := \varphi_{w_0} \wedge \bigwedge_{i\in\{0,\dots,k\}} \varphi^i_{actions}$$

with

$$\begin{aligned}\varphi^i_{actions} := {} & \varphi^i_{collectBase} \vee \varphi^i_{setupRingColor} \vee \varphi^i_{feedCap} \vee \varphi^i_{mountCap} \\ & \vee \varphi^i_{dropTransparentBase} \vee \varphi^i_{feedBase} \vee \varphi^i_{mountRing} \\ & \vee \varphi^i_{collectWorkpiece} \vee \varphi^i_{deliverWorkpiece}\end{aligned}$$

whose models represents all possible action sequences over the above mentioned actions starting at the world state w_0.

In the following sections $a \in A$ denotes the action which is discussed in the particular section.

4.2.1 Collecting a Base

The action of collecting a base with coloring $c \in C$ from a base station $bs \in S_{BS}$ can only be performed by a robot $r \in R$ if r does not hold a workpiece at the moment: $B_r^i = c_{none}$. After the action is completed, r holds a base with coloring c:
$B_r^{i+1} = c \wedge R_{1,r}^{i+1} = c_{none} \wedge R_{2,r}^{i+1} = c_{none} \wedge R_{3,r}^{i+1} = c_{none} \wedge C_r^{i+1} = c_{none}$.

The circumstance that a base stays in the output of the base station is not necessary to consider since the actions of requesting and taking the base will be encoded as a monolithic action. This is achievable without restricting the set of possible plans since a base is immediately dispensed after requested.

By taking the time aspects of the action $\varphi^i_{time(a,r,bs,out)}$, the unaffected world state properties $\varphi^i_{rem(a,r,bs)}$, and the reward representation $\varphi^i_{order(a,r,bs)}$ into account this leads to the following formula:

$$\begin{aligned}
&\varphi_{collectBase(r,bs,c)} := \\
&\quad B_r^i = c_{none} \\
&\wedge\ B_r^{i+1} = c \wedge R_{1,r}^{i+1} = c_{none} \wedge R_{2,r}^{i+1} = c_{none} \wedge R_{3,r}^{i+1} = c_{none} \wedge C_r^{i+1} = c_{none} \\
&\wedge\ \varphi^i_{time(a,r,bs,out)} \wedge \varphi^i_{rem(a,r,bs)} \wedge \varphi^i_{order(a,r,bs)}
\end{aligned}$$

4.2.2 Mounting a Cap

To mount a cap on an unfinished workpiece, three distinct actions have to be performed:

- feeding the right colored cap from the shelf into the cap station,
- free the output of the station by removing the transparent base,
- mounting the cap by feeding the workpiece into the cap station

There exist two possibilities of freeing the output of the station from the transparent base. Either a robot takes the transparent base with the purpose to feed it into a ring station or a robot takes the base and drops it on the ground immediately. This special case has to be encoded separately, since the dropping of workpieces in general will not be considered.

To denote whether a cap is already fed into a cap station and if this is the case to represent the color of the cap, variables $X^i_{ccol(cs)}$ with $cs \in S_{CS}$ are introduced.

Feed Cap For a cap station cs and a robot r which is about to feed a cap with coloring c into the cap station cs, the following preconditions have to be fulfilled:

To be able to take the transparent base with the mounted cap from the shelf, robot r needs free grippers: $B_r^i = c_{none}$; the station cs is not allowed to already have loaded a cap: $X_{ccol(cs)}^i = c_{none}$; and the cap station's output has to be empty: $B_{cs}^i = c_{none}$. Otherwise the machine is in a production state and does not accept additional orders.

In the resulting world state the cap which was dismounted from the transparent base before is loaded in the cap station: $X_{ccol(cs)}^{i+1} = c$; in the output the transparent base is located which served as carrier for the cap:
$B_{cs}^{i+1} = c_{transparent} \wedge R_{1,cs}^{i+1} = c_{none} \wedge R_{2,cs}^{i+1} = c_{none} \wedge R_{3,cs}^{i+1} = c_{none} \wedge C_{cs}^{i+1} = c_{none}$; and after the action the robot still does not hold anything:
$B_r^{i+1} = c_{none} \wedge R_{1,r}^{i+1} = c_{none} \wedge R_{2,r}^{i+1} = c_{none} \wedge R_{3,r}^{i+1} = c_{none} \wedge C_r^{i+1} = c_{none}$.

$$\begin{aligned}
&\varphi^i_{feedCap(r,cs,c)} := \\
&X_{ccol(cs)}^i = c_{none} \wedge\ B_r^i = c_{none} \wedge\ B_{rs}^i = c_{none} \\
\wedge\ &X_{ccol(cs)}^{i+1} = c \\
\wedge\ &B_{cs}^{i+1} = c_{transparent} \wedge R_{1,cs}^{i+1} = c_{none} \wedge R_{2,cs}^{i+1} = c_{none} \wedge R_{3,cs}^{i+1} = c_{none} \\
\wedge\ &C_{cs}^{i+1} = c_{none} \\
\wedge\ &B_r^{i+1} = c_{none} \wedge R_{1,r}^{i+1} = c_{none} \wedge R_{2,r}^{i+1} = c_{none} \wedge R_{3,r}^{i+1} = c_{none} \wedge C_r^{i+1} = c_{none} \\
\wedge\ &\varphi^i_{time(a,r,cs,in)} \wedge \varphi^i_{rem(a,r,cs)} \wedge \varphi^i_{order(a,r,cs)}
\end{aligned}$$

Drop Transparent Base The precondition for a cap station cs and robot r to be able to drop a transparent base from cs is that in the output of cs a transparent base is located: $B_{cs}^i = c_{transparent}$; and that the robot has free grippers: $B_r^i = c_{none}$.

As the result the cap station's output is empty: $B_{cs}^{i+1} = c_{none}$; and the robot holds nothing since it dropped the base: $B_r^{i+1} = c_{none}$.

$$\begin{aligned}
&\varphi^i_{dropTransparentBase(r,cs)} := \\
&B_{cs}^i = c_{transparent} \wedge B_r^i = c_{none} \\
\wedge\ &B_{cs}^{i+1} = c_{none} \wedge B_r^{i+1} = c_{none} \\
\wedge\ &\varphi^i_{time(a,r,cs,out)} \wedge \varphi^i_{rem(a,r,cs)} \wedge \varphi^i_{order(a,r,cs)}
\end{aligned}$$

Mount the Cap To be able to mount a previously fed cap with the aid of a cap station cs a robot r has to hold at least a base: $\neg(B_r^i = c_{none})$, which is not allowed to be transparent: $\neg(B_r^i = c_{transparent})$, and the workpiece must not have already a cap mounted: $C^i_r = c_{none}$. Beyond that, a cap has to be fed to the cap station cs previously: $\neg(X^i_{ccol(cs)} = c_{none})$ and the output has to be freed from workpieces: $B^i_{cs} = c_{none}$.

After the action is performed, the workpiece previously held by the robot is provided with a cap and is located in the output of cs:
$B^{i+1}_{cs} = B^i_r \wedge R^{i+1}_{1,cs} = R^i_{1,r} \wedge R^{i+1}_{2,cs} = R^i_{2,r} \wedge R^{i+1}_{3,r} = R^i_{3,r} \wedge C^{i+1}_r = X^i_{ccol(cs)}$.
The grippers of robot r are empty now:
$B^{i+1}_r = c_{none} \wedge R^{i+1}_{1,r} = c_{none} \wedge R^{i+1}_{2,r} = c_{none} \wedge R^{i+1}_{3,r} = c_{none} \wedge C^{i+1}_r = c_{none}$.

$$
\begin{aligned}
&\varphi^i_{mountCap(r,cs)} := \\
&\neg(B^i_r = c_{none}) \wedge \neg(B^i_r = c_{transparent}) \wedge C^i_r = c_{none} \wedge \neg(X^i_{ccol(cs)} = c_{none}) \\
&\wedge\ B^i_{cs} = c_{none} \\
&\wedge\ B^{i+1}_{cs} = B^i_r \wedge R^{i+1}_{1,cs} = R^i_{1,r} \wedge R^{i+1}_{2,cs} = R^i_{2,r} \wedge R^{i+1}_{3,r} = R^i_{3,r} \wedge C^{i+1}_r = X^i_{ccol(cs)} \\
&\wedge\ B^{i+1}_r = c_{none} \wedge R^{i+1}_{1,r} = c_{none} \wedge R^{i+1}_{2,r} = c_{none} \wedge R^{i+1}_{3,r} = c_{none} \wedge C^{i+1}_r = c_{none} \\
&\wedge\ X^{i+1}_{ccol(cs)} = c_{none} \\
&\wedge\ \varphi^i_{time(a,r,cs,in)} \wedge \varphi^i_{rem(a,r,cs)} \wedge \varphi^i_{order(a,r,cs)}
\end{aligned}
$$

4.2.3 Mounting a Ring

The mounting of a ring on a workpiece requires the execution of several distinct actions:

- specify the color of the ring a ring station will mount on the next fed workpiece,
- the actions of feeding up to two bases to a ring station if required for a particular ring color,
- mounting a ring by feeding a workpiece to a ring station,

To denote whether a ring color is already set-up and if this is the case to represent this color, variables $X^i_{rcol(rs)}$ with $rs \in S_{RS}$ are introduced. The values of the variables $X^i_{breq(rs)}$ with $rs \in S_{RS}$ indicate the remaining amount of additional bases needed to be fed in order to be able to mount the ring with the set up color.

Set Up the Ring Color Instructing a ring station rs to prepare mounting a ring with a particular coloring c does neither result in an occupation of the station nor in an occupation of the robot which gave the instruction. Therefore, the time aspect for both, the station as well as the robot, does not change in the following world state. Since the instructing robot does not have to be in a special state and after the action its state equals its previous state, the action of setting the ring color up can be seen as independent from a particular robot: the execution of the action by an arbitrary robot, only affects the ring station rs.

The ring station's output has to be empty before: $B^i_{rs} = c_{none}$; and no color has to be set up as the next mounted ring coloring: $X^i_{rcol(rs)} = c_{none}$.

After the action is performed the machine is instructed to mount a ring of color c: $X^{i+1}_{rcol(rs)} = c$; and to wait for additional bases if needed: $X^{i+1}_{breq(rs)} = b_{rs}(c)$, with $b_{rs}(c)$ denoting the amount of needed additional bases for color c.

$$\begin{aligned}
&\varphi^i_{setupRingColor(r,rs,c)} := \\
&B^i_{rs} = c_{none} \land X^i_{rcol(rs)} = c_{none} \\
\land\ &X^{i+1}_{rcol(rs)} = c \land X^{i+1}_{breq(rs)} = b_{rs}(c) \\
\land\ &\varphi^i_{rem(a,rs)}
\end{aligned}$$

Feed Base In order to be able to feed a base to a ring station rs, a robot r has to hold such, without a ring or cap mounted: $\neg(B^i_r = c_{none}) \land R^i_{1,r} = c_{none} \land C^i_r = c_{none}$; and the action has to be required by the ring station in terms of that additional bases are needed: $X^i_{breq(rs)} > 0$. Additionally the output has to be empty: $B^i_{rs} = c_{none}$.

After the base has been fed, the amount of required bases is decreased: $X^{i+1}_{breq(rs)} = X^i_{breq(rs)} - 1$, the robots grippers are empty: $B^{i+1}_r = c_{none} \land R^{i+1}_{1,r} = c_{none} \land R^{i+1}_{2,r} = c_{none} \land R^{i+1}_{3,r} = c_{none} \land C^{i+1}_r = c_{none}$.

$$\begin{aligned}
&\varphi^i_{feedBase(r,rs)} := \\
&\neg(B^i_r = c_{none}) \land R^i_{1,r} = c_{none} \land C^i_r = c_{none} \land X^i_{breq(rs)} > 0 \land B^i_{rs} = c_{none} \\
\land\ &X^{i+1}_{breq(rs)} = X^i_{breq(rs)} - 1 \\
\land\ &B^{i+1}_r = c_{none} \land R^{i+1}_{1,r} = c_{none} \land R^{i+1}_{2,r} = c_{none} \land R^{i+1}_{3,r} = c_{none} \land C^{i+1}_r = c_{none} \\
\land\ &\varphi^i_{time(a,r,rs,in)} \land \varphi^i_{rem(a,r,rs)} \land \varphi^i_{order(a,r,rs)}
\end{aligned}$$

Mount the Ring Given that a workpiece is fed by a robot r into the ring station rs in order to mount a previously set-up ring, the workpiece has to fulfill several preliminaries. The workpiece has to exist: $\neg(B^i_r = c_{none})$; the base is not allowed to be transparent: $\neg(B^i_r = c_{transparent})$; besides that, there has to be space for another ring, in other words the third ring is not mounted: $R^i_{3,r} = c_{none}$. Further it has to be considered that a ring cannot be mounted if a cap is already mounted: $C^i_r = c_{none}$. Additionally the output has to be free: $B^i_{rs} = c_{none}$; a ring color has to be set up: $\neg(X^i_{rcol(rs)} = c_{none})$; and possible requirements for additional bases are fulfilled: $X^i_{breq(rs)} = 0$. As a result, the ring station is no longer instructed to mount a ring of a particular color: $X^{i+1}_{rcol(rs)} = c_{none}$; and the robot is no longer holding the workpiece:

$$B^{i+1}_r = c_{none} \land R^{i+1}_{1,r} = c_{none} \land R^{i+1}_{2,r} = c_{none} \land R^{i+1}_{3,r} = c_{none} \land C^{i+1}_r = c_{none}$$

Now the workpiece the robot previously held by the robot is located in the output of the ring station and mounted with an additional ring. The color of the base is maintained $B^{i+1}_{rs} = B^i_r$, while a cap still is absent $C^{i+1}_{rs} = c_{none}$. In consideration of the encoding's representation of workpieces, a distinction has to be made, whether the first, the second or the third ring will be mounted. In the following, formulas φ_{ring_1}, φ_{ring_2} and φ_{ring_3} will be derived, each expresses that the corresponding positioned ring is mounted:

In case that the workpiece has no ring mounted the constraint $R^i_{1,r} = c_{none}$ holds, in the representation of the following world state, the set-up color $X^i_{rcol(rs)}$ will be assigned to the variable $R^{i+1}_{1,rs}$ expressing the color of the first ring in the following world state, while the remaining ring-variables still represent the nonexistence of their associated rings:
$R^{i+1}_{1,rs} = X^i_{rcol(rs)} \land R^{i+1}_{2,rs} = c_{none} \land R^{i+1}_{3,rs} = c_{none}$.

$$\begin{aligned}\varphi_{ring_1} := {} & R^i_{1,r} = c_{none} \\ & \land R^{i+1}_{1,rs} = X^i_{rcol(rs)} \land R^{i+1}_{2,rs} = c_{none} \land R^{i+1}_{3,rs} = c_{none}\end{aligned}$$

If the first ring is already mounted $\neg(R^i_{1,r} = c_{none})$ but the second ring is not mounted yet $R^i_{2,r} = c_{none}$, the following world state maintains the coloring of the first ring $R^{i+1}_{1,rs} = R^i_{1,r}$ and places the set-up color of rs as the coloring of the second ring $R^{i+1}_{2,rs} = X^i_{rcol(rs)}$, while $R^{i+1}_{3,rs} = c_{none}$ still represents the nonexistence of the third ring.

$$\begin{aligned}\varphi_{ring_2} := {} & \neg(R^i_{1,r} = c_{none}) \land R^i_{2,r} = c_{none} \\ & \land R^{i+1}_{1,rs} = R^i_{1,r} \land R^{i+1}_{2,rs} = X^i_{rcol(rs)} \land R^{i+1}_{3,rs} = c_{none}\end{aligned}$$

In case that the first and second ring are mounted, it is sufficient to check the constraint $\neg(R^i_{2,r} = c_{none})$ since the structure of the formulas ensures that if a particular color is assigned to the variable $R^i_{2,r}$ the variable $R^i_{1,r}$ has also a color assigned to it. If furthermore the third ring is not mounted $R^i_{3,r} = c_{none}$ the third ring will be mounted $R^{i+1}_{3,rs} = X^i_{rcol(rs)}$ while the first and second ring maintain their coloring $R^{i+1}_{1,rs} = R^i_{1,r} \land R^{i+1}_{2,rs} = R^i_{2,r}$.

$$\begin{aligned}\varphi_{ring_3} := {} & \neg(R^i_{2,r} = c_{none}) \land R^i_{3,r} = c_{none} \\ & \land R^{i+1}_{1,rs} = R^i_{1,r} \land R^{i+1}_{2,rs} = R^i_{2,r} \land R^{i+1}_{3,rs} = X^i_{rcol(rs)}\end{aligned}$$

Since one of the formulas has to be satisfied if a ring is mounted in the ith action, their disjunction $\varphi_{ring_1} \vee \varphi_{ring_2} \vee \varphi_{ring_3}$ covers all cases where a ring is mountable and expresses the preconditions and results of this action. Therefore the following formula represents the process of mounting a ring on the first, second or third position:

$$\varphi_{mount} := B^{i+1}_{rs} = B^i_r \wedge (\varphi_{ring_1} \vee \varphi_{ring_2} \vee \varphi_{ring_3}) \wedge C^{i+1}_{rs} = c_{none}$$

This leads to the following formula expressing the whole process of mounting a ring:

$$\begin{aligned}
&\varphi^i_{mountRing(r,rs)} := \\
&\neg(B^i_r = c_{none}) \wedge \neg(B^i_r = c_{transparent}) \wedge C^i_r = c_{none} \wedge R^i_{3,r} = c_{none} \wedge B^i_{rs} = c_{none} \\
&\wedge \neg(X^i_{rcol(rs)} = c_{none}) \wedge X^i_{breq(rs)} = 0 \\
&\wedge B^{i+1}_r = c_{none} \wedge R^{i+1}_{1,r} = c_{none} \wedge R^{i+1}_{2,r} = c_{none} \wedge R^{i+1}_{3,r} = c_{none} \wedge C^{i+1}_r = c_{none} \\
&\wedge X^{i+1}_{rcol(rs)} = c_{none} \wedge X^{i+1}_{breq(rs)} = 0 \\
&\wedge \varphi_{mount} \\
&\wedge \varphi^i_{time(a,r,rs,in)} \wedge \varphi^i_{rem(a,r,rs)} \wedge \varphi^i_{order(a,r,rs)}
\end{aligned}$$

4.2.4 Pick up Workpiece

Since only the ring and cap stations are considered to be capable of holding workpieces, only for these, formulas have to be created which express the collecting of workpieces.

In the output of a ring or cap station s a workpiece has to be located: $\neg(B^i_s = c_{none})$, a robot r can pick it up only if its grippers are empty: $B^i_r = c_{none}$. After the action, the workpiece is exchanged from the station s to the robot: $B^{i+1}_r = B^i_s \wedge R^{i+1}_{1,r} = R^i_{1,s} \wedge R^{i+1}_{2,r} = R^i_{2,s} \wedge R^{i+1}_{3,r} = R^i_{3,s} \wedge C^{i+1}_r = C^i_s$. Therefore the station s holds nothing in the following world state: $B^{i+1}_s = c_{none} \wedge R^{i+1}_{1,s} = c_{none} \wedge R^{i+1}_{2,s} = c_{none} \wedge R^{i+1}_{3,s} = c_{none} \wedge C^{i+1}_s = c_{none}$.

$$\begin{aligned}
&\varphi^i_{collectWorkpiece(r,s)} := \\
&B^i_r = c_{none} \wedge \neg(B^i_s = c_{none}) \\
&\wedge B^{i+1}_s = c_{none} \wedge R^{i+1}_{1,s} = c_{none} \wedge R^{i+1}_{2,s} = c_{none} \wedge R^{i+1}_{3,s} = c_{none} \wedge C^{i+1}_s = c_{none} \\
&\wedge B^{i+1}_r = B^i_s \wedge R^{i+1}_{1,r} = R^i_{1,s} \wedge R^{i+1}_{2,r} = R^i_{2,s} \wedge R^{i+1}_{3,r} = R^i_{3,s} \wedge C^{i+1}_r = C^i_s \\
&\wedge \varphi^i_{time(a,r,s,out)} \wedge \varphi^i_{rem(a,r,s)} \wedge \varphi^i_{order(a,r,s)}
\end{aligned}$$

4.2.5 Deliver Workpiece

Given a robot r and a delivery station ds, a finished product is composed of at least a base and a cap. It suffices to check for the existence of the cap:

$\neg(C_r^i = c_{none})$, if the cap is present, the existence of the base follows in the encoding. After delivering the product, the robot holds nothing:
$B_r^{i+1} = c_{none} \wedge R_{1,r}^{i+1} = c_{none} \wedge R_{2,r}^{i+1} = c_{none} \wedge R_{3,r}^{i+1} = c_{none} \wedge C_r^{i+1} = c_{none}$.

$$\begin{aligned}
&\varphi^i_{deliverWorkpiece(r,ds)} := \\
&\neg(C_r^i = c_{none}) \\
&\wedge B_r^{i+1} = c_{none} \wedge R_{1,r}^{i+1} = c_{none} \wedge R_{2,r}^{i+1} = c_{none} \wedge R_{3,r}^{i+1} = c_{none} \wedge C_r^{i+1} = c_{none} \\
&\wedge \varphi^i_{time(a,r,s,in)} \wedge \varphi^i_{rem(a,r,s)} \wedge \varphi^i_{order(a,r,s)}
\end{aligned}$$

It has to be mentioned that the delivery of a product has to be seen as a distinct action, since it is not sufficient to assume that after the completion of an product the robot which initiated the mounting of the cap also delivers the product. In this case the robot would have to wait for the completion of the product during which it could perform other actions. Therefore, this simplification would restrict the set of possible plans.

4.3 The Optimal Action Sequence

The goal of the RoboCup Logistic League Game is to get the highest possible score by producing and delivering products. That means an action sequence has to be found which maximizes the sum of the overall earned reward. It has to be mentioned that an action sequence is seen as optimal in regards to the gained points and not an efficient time use. In the following section, first, an encoding is derived tracking whether an order is delivered. This enables the optimization in regards to a variable representing the gained reward by means of an optimizing SMT-solver, leading to the highest rewarded realizable action sequence. Whereas the secondly presented encoding tracks all steps of the order progressions, this results in only allowing action sequences which work towards finishing all orders and therefore to the highest possible reward, not considering the time limits. This approach supersedes the use of an *optimizing* SMT-solver.

4.3.1 Tracking the Order Delivery

To be able to utilize a optimizing SMT-solver to find an optimal action sequence, the gained points have to be tracked in the world state representation. This enables the optimization towards a term representing the overall earned reward in the last world state.

Reward Variable A variable X^i_{reward} denoting the reward gained until reaching the world state on position $i \in \{0, ..., k+1\}$ is introduced. That means, X^{k+1}_{reward} represents the reward gained after performing the action sequence on the initial state. Assume the gained reward is tracked by a formula $\varphi^i_{reward(a,r,s)}$ in a way that only the complexity of the processing step is taken into account, but the processing of the orders is not checked; for the action $a \in A$, the involved robot $r \in R$, and the used station $s \in S$. Using

$$\varphi^i_{order(a,r,s)} := \varphi^i_{reward(a,r,s)}$$

as the only representation of the gained reward violates the condition that multiple production steps for the same order must not be rewarded multiple times. By searching for an action sequence maximizing X^{k+1}_{reward}, the processing step which is easiest to perform and gives the most reward, relative to the needed steps, is performed as often as possible. Especially orders may be served multiple times.

Tracking the Order Delivery The problem mentioned in the previous chapter, that products are delivered more often than ordered can be solved by adding a boolean variable $X^i_{delivered(o)}$ whose satisfaction indicates that the product required by the order $o \in O$ is already delivered.

To achieve the mentioned properties, the formula $\varphi^i_{order(a,r,ds)}$ is defined in the following way for the action-formulas representing the delivery of a product by robot $r \in R$: Let $o \in O$ be an arbitrary order. By fulfilling

$$\begin{aligned} B^i_r = c_{base}(o) \wedge R^i_{1,r} = c_{ring,1}(o) \wedge R^i_{2,r} = c_{ring,2}(o) \\ \wedge\ R^i_{3,r} = c_{ring,3}(o) \wedge C^i_r = c_{cap}(o) \end{aligned}$$

it is ensured that the delivered product matches the requirements of the order o. Let $r_{deadline} : O \mapsto \mathbb{N}$ be the function which maps the order o to the point in time until which o has to be served. It follows, that the constraint

$$X^{i+1}_{occ(ds)} < r_{deadline}(o)$$

ensures that only timely deliveries are performed. By checking the fulfillment of $\neg X^i_{delivered(o)}$, it is assured that the order o is not served yet in the world state the action is performed in, while forcing $X^{i+1}_{delivered(o)}$ to be fulfilled represents that the delivery of the product for order o is already performed in the following world state. To guarantee the consistent representation of the remaining orders, their status has to be inherited:

$$\bigwedge_{o' \in O \setminus \{o\}} X^i_{delivered(o')} \to X^{i+1}_{delivered(o')}$$

These constraints also have the effect that only one order at a time can be delivered. Further let $\varphi^i_{reward(a,r,s)}$ be the formula which enables the tracking of the gained points in a world state, as described in the previous section.

The above mentioned assumptions lead to the following definition of the formula $\varphi^i_{order(a,r,ds)}$ for the action-formulas $\varphi^i_{deliverWorkpiece(r,ds)}$, with $a \in A$ the delivery-action, $ds \in S_{DS}$ a delivery station, and $r \in R$ a robot:

$$\begin{aligned}
&\varphi^i_{order(a,r,ds)} := \\
&\quad \bigvee_{o \in O} (B^i_r = c_{base}(o) \wedge R^i_{1,r} = c_{ring,1}(o) \wedge R^i_{2,r} = c_{ring,2}(o) \\
&\qquad \wedge R^i_{3,r} = c_{ring,3}(o) \wedge C^i_r = r_{cap}(o) \\
&\qquad \wedge X^{i+1}_{occ(ds)} < r_{deadline}(o) \wedge \neg X^i_{delivered(o)} \wedge X^{i+1}_{delivered(o)} \\
&\quad \bigwedge_{o' \in O \setminus \{o\}} X^i_{delivered(o')} \rightarrow X^{i+1}_{delivered(o')} \\
&\qquad \wedge \varphi^i_{reward(a,r,ds)})
\end{aligned}$$

For all remaining actions $a' \in A \setminus \{a\}$ and the used stations corresponding $s' \in S \setminus S_{DS}$ the formula $\varphi^i_{order(a',r,s')}$ is defined as known from the previous section:

$$\varphi^i_{order(a',r,s')} := \varphi^i_{reward(a',r,s')}$$

This adaption will solve the problem only partly, a product will be delivered only once to fulfill an orders requirement, but unnecessary steps will still be performed and rewarded. Therefore, the model which maximizes X^{k+1}_{reward} will not represent an action sequence leading to the highest reward.

By choosing the action sequences length high enough, this solution will find at least an action sequence which serves a set of orders if this is possible regarding the time constraints. The delivery of a product is the only encoded possibility for a robot to free its hands besides feeding a workpiece into a station. If this possibility has been exhausted the robot is forced to deliver the product. On the other hand, by choosing the action sequences length too high, the formula resulting from this "solution" will be unsatisfiable. After serving the set of orders which requires the highest amounts of actions, the robots fill their hands and stations with workpieces until no other action is possible.

One may take into account to expand the action-formulas to make them only satisfiable if an order exists whose required product needs the action and which is not yet delivered in the world state the action is performed in. This does not solve the problem that processing steps for an ordered product can be performed multiple times. If the order is not delivered, the steps which were performed for the order before can be still performed again.

Not Rewarding Intermediate Steps By rewarding only the delivery of a product it is avoided that the tracking of gained points is distorted by the rewarding of duplicate processing steps. Useless actions may still occur in the action sequence, but can be identified and ignored when the action plan is executed. Therefore an optimal action sequence, without considering the useless intermediate steps, will be found if the delivery is rewarded by the sum of the points gained during the whole production process and if the length of the action sequence is chosen high enough that all possibilities of serving the orders are considered. On the other hand φ may be still unsatisfiable if the length of the action sequence is chosen too high. This problem can be solved by not forbidding the delivery of a product if the deadline for an order is expired or all orders demanding this product are already served, instead do not reward the delivery in these cases.

In the following let $r_{reward} : O \mapsto \mathbb{N}$ denote the sum of points awarded for producing and delivering an ordered product.

It has to be expressed that under the circumstance that the deadline for an order $o \in O$ is expired or o is already served, no reward is awarded:

$$(X^{i+1}_{occ(ds)} \geq r_{deadline}(o) \vee X^{i}_{delivered(o)}) \wedge (X^{i+1}_{delivered(o)} \wedge X^{i+1}_{reward} = X^{i}_{reward})$$

Is the product delivered in time and o is not served yet, o is marked as delivered in the following world state and the points for producing the products are awarded:

$$\begin{aligned}&(X^{i+1}_{occ(ds)} < r_{deadline}(o) \wedge \neg X^{i}_{delivered(o)})\\ \wedge\ &(X^{i+1}_{delivered(o)} \wedge X^{i+1}_{reward} = X^{i}_{reward} + r_{reward}(o))\end{aligned}$$

The above mentioned observations lead to a redefinition of the formula

$$\begin{aligned}
&\varphi^{i}_{order(a,r,ds)} := \\
&\bigvee_{o \in O} (B^{i}_{r} = r_{base}(o) \wedge R^{i}_{1,r} = r_{ring,1}(o) \\
&\quad \wedge\ R^{i}_{2,r} = r_{ring,2}(o) \wedge R^{i}_{3,r} = r_{ring,3}(o) \wedge C^{i}_{r} = r_{cap}(o) \\
&\quad \wedge\ ((X^{i+1}_{occ(ds)} \geq r_{deadline}(o) \vee X^{i}_{delivered(o)}) \\
&\qquad\qquad \wedge (X^{i+1}_{delivered(o)} \wedge X^{i+1}_{reward} = X^{i}_{reward}) \\
&\quad \vee\ (X^{i+1}_{occ(ds)} < r_{deadline}(o) \wedge \neg X^{i}_{delivered(o)}) \\
&\qquad\qquad \wedge (X^{i+1}_{delivered(o)} \wedge X^{i+1}_{reward} = X^{i}_{reward} + r_{reward}(o))) \\
&\bigwedge_{o' \in O \setminus \{o\}} X^{i}_{delivered(o')} \rightarrow X^{i+1}_{delivered(o')})
\end{aligned}$$

with $a \in A$ the delivery-action, $ds \in S_{DS}$ a delivery station, and $r \in R$ a robot. Since all remaining actions $a' \in A \setminus \{a\}$ on the used corresponding stations $s' \in S \setminus S_{DS}$ are not rewarded the formula $\varphi^i_{order(a',r,s')}$ is defined in the following way:

$$\varphi^i_{order(a',r,s')} := X^{i+1}_{reward} = X^i_{reward}$$

These changes enable the derivation of an optimal action sequence by using an optimizing SMT-solver to find a model of φ maximizing the term X^{k+1}_{reward}.

A drawback is that it is needed to look at complete action sequences, this means the last action for a particular product is a delivery action, incomplete action sequences will lower the reward in the end since only the delivery of the product is rewarded positively. Therefore to consider all orders a formula has to be created which covers an action sequence with a length which allows theoretically the production of all ordered products, not considering the possible restrictions by the deadlines.

The Length of the Action Sequence In the following a procedure is derived to compute the necessary length of the action sequences to enable all possibilities of serving the set of orders O. The only uncertainty when computing the length for the optimal action sequences is, whether a transparent base is dropped from a cap station's output or collected to be fed into a ring station. The first case requires the actions of dropping the transparent base and additionally the collecting of a base from a base station while the second case requires only the action of collecting the transparent base to make the cap station usable again and to be able to feed the base into a ring station. In the following it is described which actions have to be performed to produce a product:

- The base of the product needs to be collected.
- To mount a ring:
  - the color of the ring has to be set up,
  - depending on the coloring of the ring, zero to two bases have to be collected and fed into a ring station,
  - if the ring is mounted on the second or third position, the workpiece has to be taken from the output of a ring station,
  - the workpiece has to be fed into the ring station.
- To mount a cap:

  - a cap has to be fed into the cap station,
  - *if not used to feed a ring station the remaining transparent base has to be dropped*,
  - if the product has rings mounted the workpiece has to be taken from a ring station,
  - the workpiece has to be fed into the cap station.

- To deliver the product:
  - the product has to be taken from the output of the cap station,
  - the product has to be fed into the delivery station.

One can derive that six steps are needed to collect the base, mount a cap and deliver the product. Three additional steps are required to mount a ring. Two additional steps are needed for each additional base required. This leads to the following formula which determines the maximum steps required to produce a product: $req\_steps = 6 + 3 * r + 2 * b$ with b the overall amount of additional bases needed and r the amount of rings mounted on the product.

This information allows the derivation of the maximum needed steps to serve all orders in O by summing up the maximum needed steps for producing the products demanded by all $o \in O$.

Speeding up the Optimizer by Providing Information Explicitly In order to speed up the solving process, additional constraints can be added expressing implicit encoded information explicitly. This allows the solver to detect unsatisfiable instances easier.

The overall gained reward in a step is always smaller than or equal to the overall gained possible reward:

$$\bigwedge_{i \in \{1,\ldots,k\}} X^i_{reward} \leq \sum_{o \in O} r_{reward(o)}$$

This information allows the optimizer to stop after an action sequence is found which serves all orders.

Idle Action and Negatively Rewarded Intermediate Steps To avoid the possibility that unnecessary actions occur in the action sequence one may introduce another action-formula representing the circumstance that nothing changes from a world state to the following world state; all variable values are inherited from the previous world state. By rewarding this action with zero points and the intermediate steps with a negative value the optimizer will avoid the intermediate actions unless they lead to a overall positive reward, instead the "do-nothing"-action will be chosen, not lowering the reward. Introduced is a new boolean variable $X^i_{doNothing}$ to indicate, that the "do-nothing"-action was performed in step i, it is encoded that when $X^i_{doNothing}$ is set, only the "do-nothing"-action is allowed in step $i+1$. This approach allows only "do-nothing"-actions on the end of the action sequence, otherwise the optimizing time would be increased unnecessarily by this alteration, since the amount of potential action sequences would be highly increased.

To guarantee that this solution still leads to an optimal action sequence the following has to hold:

- the reward awarded for the serving of the orders has to be chosen in a way that the sum of the gained reward for performing the intermediate steps and the reward for the delivery is greater than zero,
- the negative reward of the intermediate steps has to have no impact on the rating of the different products.

A product with three rings mounted which need two additional bases each, needs 26 or 27 steps to be produced and delivered if it is the only ordered product. That also means that 27 is the maximum amount of steps needed to produce any single product. Therefore, the above mentioned properties are fulfilled by choosing the reward for serving an order as the sum of all points gained during the production process multiplied by 100. The only change in the rating is that the product which needs a lower amount of actions is preferred out of two products whose reward was previously the same.

4.3.2 Tracking All Steps of the Order Progression

A drawback of the earlier mentioned encoding is the outstandingly higher solving time when searching for an optimized model, rather than an arbitrary model. The following encoding restricts the set of models of φ to assignments representing only sequences of actions which avoid duplicate production steps by tracking the whole order progress. That is, instead of distinguishing solely between delivered and non-delivered, order states will be defined corresponding to the actions. In this way for each action-formula constraints can be created which check if the particular action is needed by an order.

Order Progressions The following order progressions can be distinguished:

- the base for the product is collected,

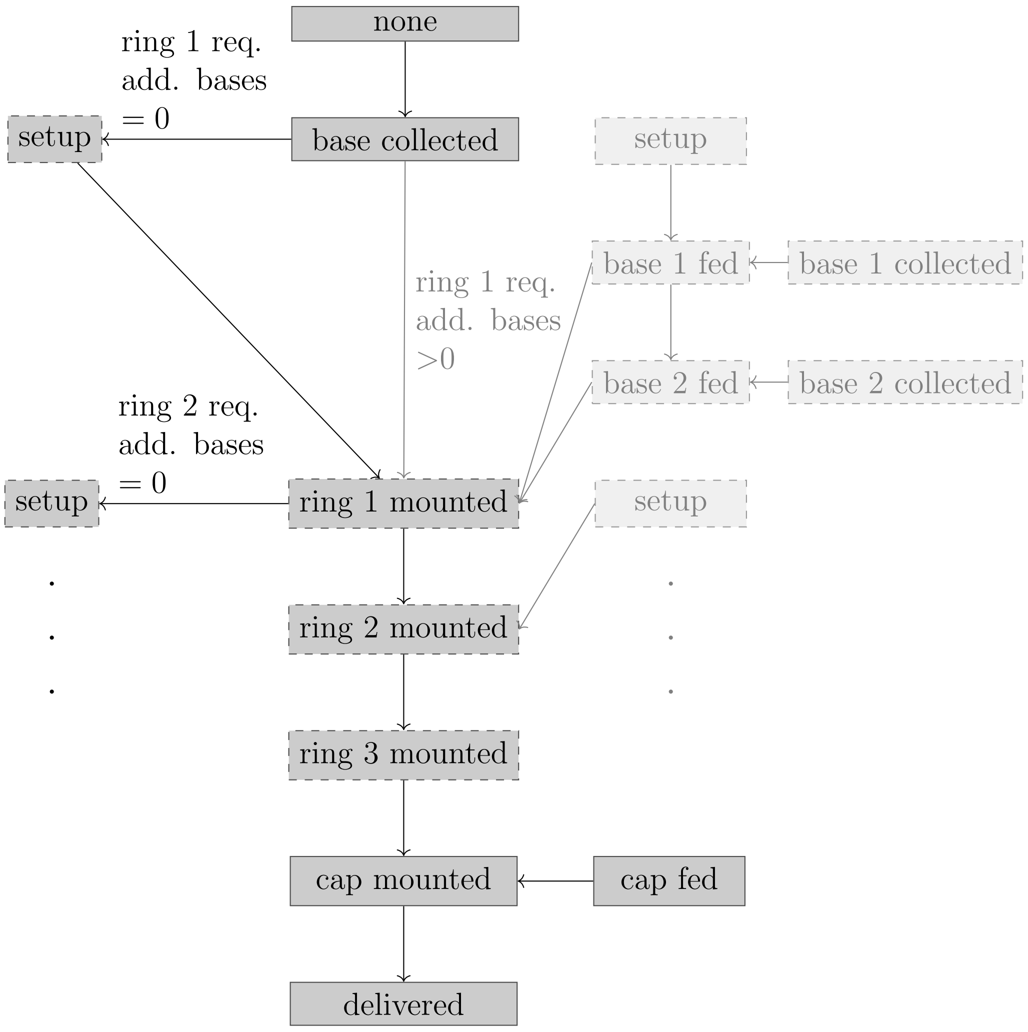


Figure 7: Order progress

- if required, the coloring of the ring on the first/second/third position is set up in a ring station,
- if required, the first/second additional base for the ring on the first/second/third position is collected,
- if required, the first/second additional base for the ring on the first/second/third position is fed into the ring station,
- the ring on the first/second/third position is mounted,
- the cap is fed into the cap station,
- the cap is mounted,
- the product is delivered.

The order these progression steps are performed in is not fixed. That means the coloring of a ring can be set up even before the base is collected, the same applies to the collecting and feeding of an additional base and the feeding of a cap. Likewise, the additional bases for a ring color can be collected before the ring station is set up for this color; and the second additional base for a ring color can be collected at the same time the first additional base is collected.

Some progression steps are dependent on other progression steps: The mounting of the first ring requires that the base is collected; the mounting of the second and third ring requires that the ring on the previous position is mounted; and the mounting of the cap requires that the last ring, if present, is mounted or, if no ring is present, the base is collected. Furthermore the feeding of an additional base requires the setup of the ring color and the collecting of the base. Additionally, the mounting of a ring requires, if no additional bases are needed, that the particular color is set up, or that the amount of needed bases are fed. Whereas the setting up of a ring color and the collecting of the first and second additional base have no requirements.

Figure 7 pictures these relations.

In order to respect the above mentioned circumstances for each order $o \in O$ one variable for each of those order progression stages which are capable of being performed in parallel, is created.

Order Progression Variables

- The value $v \in \{s_{none}, s_{base}, s_{ring_1}, s_{ring_2}, s_{ring_3}, s_{cap}, s_{delivered}\} \subseteq N$ assigned to the integer variable $X^i_{main(o)}$ denotes which component of the product order o demands was mounted last.

- The boolean variable $X^i_{setup(j,o)}$ indicates whether the color for the ring on position $j \in \{1,2,3\}$ is set up.

- The value $v \in \{s_{none}, s_{collected}, s_{fed}\} \subseteq N$ assigned to the integer variable $X^i_{addBase(j,l,o)}$ indicates whether the lth additional base for the ring on position j is collected or fed, with $j \in \{1,2,3\}$ and $l \in \{1,2\}$.

- The boolean variable $X^i_{cap(o)}$ indicates whether the cap for the ordered product is already fed into the cap station.

for all world state positions $i \in \{0,..,k+1\}$.

By means of these variables it is possible to create a formula to check for each action, except the actions collect-workpiece and drop-transparent-base, if their performing is necessary to produce an ordered product.

In the following the exemplary construction of the formulas $\varphi^i_{order(a,r,cs)}$ for the mount-cap-action formulas $\varphi^i_{mountCap(r,cs)}$ is shown, with $f_{prevState}(o, p_{cap})$ denoting the id of the order state prior to the state represented by p_{cap} of order o, $a \in A$ the mount-cap action, $r \in R$ the involved robot, and $cs \in S_{CS}$ the operated cap station:

$$\begin{aligned}
&\varphi^i_{order(a,r,cs)} := \\
&\qquad \bigvee_{o \in O} B^i_r = r_{base}(o) \wedge R^i_{1,r} = r_{ring,1}(o) \wedge R^i_{2,r} = r_{ring,2}(o) \\
&\qquad \wedge\, R^i_{3,r} = r_{ring,3}(o) \wedge X^{i+1}_{ccol(cs)} = r_{cap}(o) \\
&\quad \wedge\, X^i_{main(o)} = f_{prevState}(o, p_{cap}) \wedge X^i_{cap(o)} \wedge X^{i+1}_{main(o)} = p_{cap}
\end{aligned}$$

Idle Action Let $o \in O$ be the only announced order, further let $k+1$ be set to the maximum number of actions needed to serve o. In the case that the action sequence using the transparent base is realizable while the other using a base from the base station is unsatisfiable, no model for φ exists. This results from the fact that only sequences with $k+1$ actions are represented by models of φ, but if a transparent base is used the sequence has k actions. To avoid this problem an additional action has to be introduced, representing the end of the plan if all orders are delivered and allowing action sequences smaller than $k+1$. This is realized by demanding for the precondition of the action, that all orders are delivered and by inheriting the delivery status to the resulting world state; the other variables are not set to a particular value. As a result the first idle action denotes the end of the plan and the following world states are undefined.

Problem: Unsatisfiable φ Let $k+1$ be set to the amount of steps needed to serve all open orders. It follows the unsatisfiability of φ, if no realizable action sequence exists which serves all orders in the time limit. In this case the solving time is usually significantly higher, it follows that the robot has to wait for the result of the solver, and eventually does not even get an instruction. To address this problem, the following approaches are possible:

Start by determining a scheduling considering only a subset of the orders, such that their fulfillment is definitely achievable and increase the amount of orders until a solving time threshold is reached.

A variation to the above mentioned approach is to run the solving processes in parallel. In this way the circumstance that each robot is equipped with a powerful processor can be exploited.

4.4 Action Translation

To translate a model of φ to the corresponding action sequence, pairs of consecutive world states have to be examined to derive the corresponding action on a particular position in the sequence.

In this section the exemplary identification of a mount-cap action on position $i \in \{0, ..., k\}$ is shown:

An assignment representing an action sequence which states that a robot $r \in R$ performs the mount-cap action on the cap station $cs \in S_{CS}$ models also the following formula:

$$\begin{gathered} \neg(B_r^i = c_{none}) \wedge B_r^{i+1} = c_{none} \\ \wedge \neg(X_{ccol(cs)}^i = c_{none}) \wedge X_{ccol(cs)}^{i+1} = c_{none} \end{gathered}$$

The mount-cap action and the involved cap station cs are uniquely identified by its precondition: $\neg(X_{ccol(cs)}^i = c_{none})$, and the alteration it induces on the world state: $X_{ccol(cs)}^{i+1} = c_{none}$. While the performing robot is determined by comparing the status of its grippers before and after the action:
$\neg(B_r^i = c_{none}) \wedge B_r^{i+1} = c_{none}$. The remaining actions can be identified in the same way.

4.5 Benchmark

The following section presents benchmarks of implementations of the two different approaches described in Chapters 4.3.1 and 4.3.2. Additionally the variant of the approach presented in Chapter 4.3.1 adding the idle-action to the set of possible actions is tested.

All experiments were performed on an Intel® Xeon(R) CPU E3-1230 v3 at 3.30GHz with 16 GB RAM with Fedora Linux using the Z3 solver version 4.5.0 [dMB08] and its optimizing functionality Z3Opt [BPF15].

Plans for several environments and orders which have to be produced and delivered in the different environments will be computed and the time needed to find a model to the corresponding formula is measured. The environment is always set up in the following way:
On the field there are one base station which is able to dispense red, black and silver colored bases, two cap stations responsible for mounting black or grey caps, two ring stations able to mount green and orange respectively blue and yellow colored rings, and a delivery station. To mount an orange ring, two additional bases are needed, a green ring needs one additional base, and yellow and blue rings need no additional base. The processing times for the stations are the means of the times from Table 1. The only alteration in the environment setup are the travel times between the robots in the starting zone and the stations, and the travel times between the stations. This data is gained from random scenarios from the simulation environment of the RoboCup Logistics League [NKVT16]. Since the data is represented as travel distances it has to be converted to travel times, thereby a travel speed of 0.1 meters per second is assumed. The travel times are represented in milliseconds and are encoded as integer numbers.

Strings of the following structure are used to denote the product an order demands: The first letter denotes the base color, the last letter the cap color, and all letters in between the ring colors. The component colors black and blue are denoted by the letter 'B', green and grey by the letter 'G', yellow by the letter 'Y', red by the letter 'R', orange by the letter 'O' and silver by the letter 'S'. Since only caps can be colored black or grey and only rings can be colored blue or green the notation is unambiguous. If the planning for two or more orders is tested this is denoted by the product notation combined with a '+'.

All experiments are performed with the maximum needed number of steps to fulfill all orders. If not stated otherwise, an order combination is tested in 10 different environments.

4.5.1 Tracking the Order Delivery

The Figures 8 and 9 show the needed solving times to compute an action plan for several orders in 10 different environments. It can be seen that the solving time rises, the greater the complexity of the ordered products is. This can be explained by the increasing length of the formula, since for more complex products more steps are needed. The wide range of the solving times for the same product in different environments can be explained by the circumstance that in some environments more action sequences maximizing the score exist, making the finding of such easier for the optimizer; since the reward threshold is encoded explicitly the solver then stops.

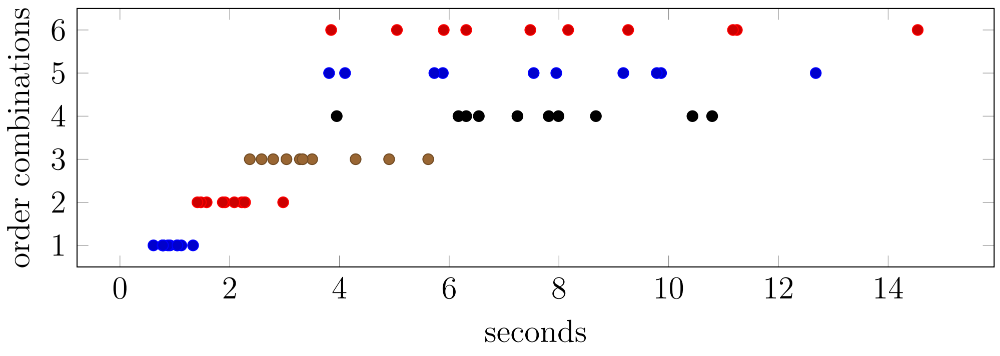


Figure 8: 1:BG, 2:BBG, 3:BGG, 4:BBBG, 5:BOG, 6:RG+RG+RG, all with 900 sec deadline

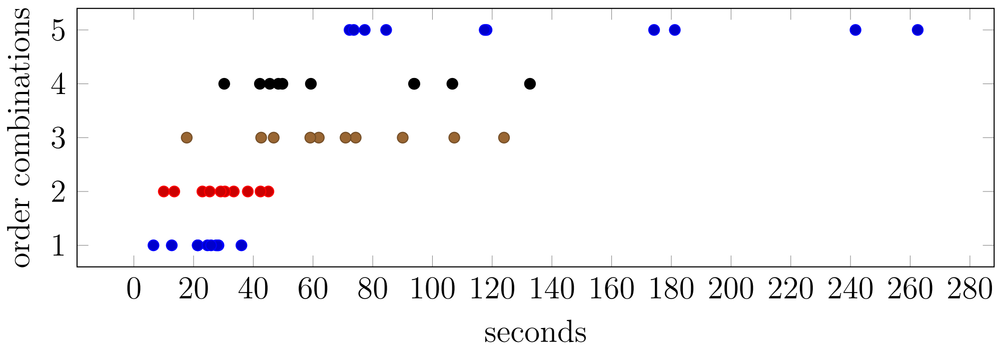


Figure 9: 1:BBBBG, 2:BGGG, 3:BOOG, 4:SBG+SGG, 5:BGGGG, all with 900 sec deadline

The above mentioned assumption is supported by Figure 10, since the solving time of order combinations with deadlines of 900 seconds is much smaller than the solving time for the same order combination with lower deadlines. The remaining solving time distribution shown in Figure 10 may be explained by the circumstance that deadlines which are so tight that the non-fulfillment of orders can be foreseen fast by the solver, which results in smaller solving times; while deadlines allowing the serving of orders lead to higher solving times.

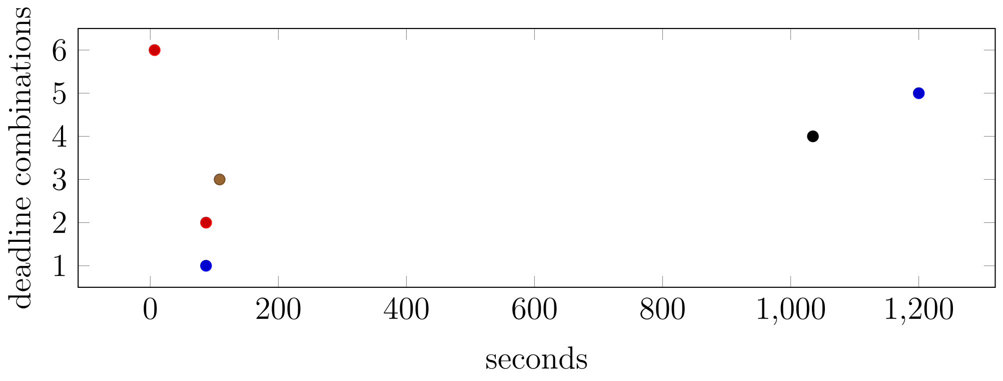


Figure 10: RG+RG+RG in the same environment, with 1: 200sec, 200sec, 200sec deadlines, no order served; 2: 500sec, 200sec, 200sec deadlines, one order served; 3: 300sec, 200sec, 200sec deadlines, one order served; 4: 500sec, 500sec, 200sec deadlines, two orders served; 5: 500sec, 500sec, 500sec deadlines, all orders served; 6: 900sec, 900sec, 900sec deadlines, all orders served

The formulas for the orders SBG+SGG+SOG, SG+SBG+SGG, and SOG+RG+BBBG were not solvable within the time period of 20 minutes.

4.5.2 Tracking the Order Delivery and Idle Action

Figure 11 shows the increasing solving time when adding an idle-action. Even the restriction to action sequences performing the idle-action at the end increases the solving time significantly.

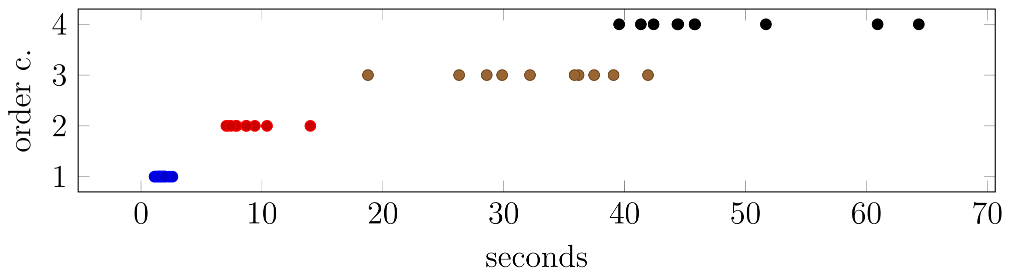


Figure 11: 1:BG, 1:BBG, 3:BGG, 4:BOG, all with 900 sec deadline

4.5.3 Tracking All Steps of the Order Progression

Figure 12 shows significantly decreased solving times when using the encoding explained in Chapter 4.3.2 compared to the previous encoding. The encoding restricts the set of action sequences represented by the formulas to those which serve the orders, therefore it is sufficient to search for arbitrary models instead of an optimized one. Since deadlines of 900 seconds are used the amount of possible action sequences is quite high and the solver can easily find a model. Just like the other encoding, the solving time rises with order complexity and thus with raising variable count and formula length. This can be seen in the Figures 12, 13, 14, and 15.

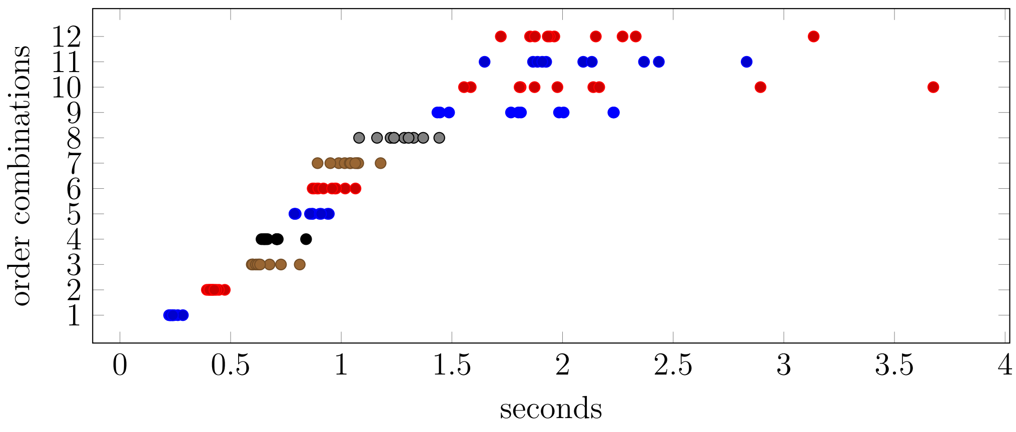


Figure 12: 1:BG, 1:BBG, 3:BGG, 4:BBBG, 5:BOG, 6:BBBBG, 7:BGGG, 8:RG+RG+RG, 9:BGGGG, 10:BOOG, 11:SBG+SGG, 12:BOGBG, all with 900 sec deadline and all satisfiable

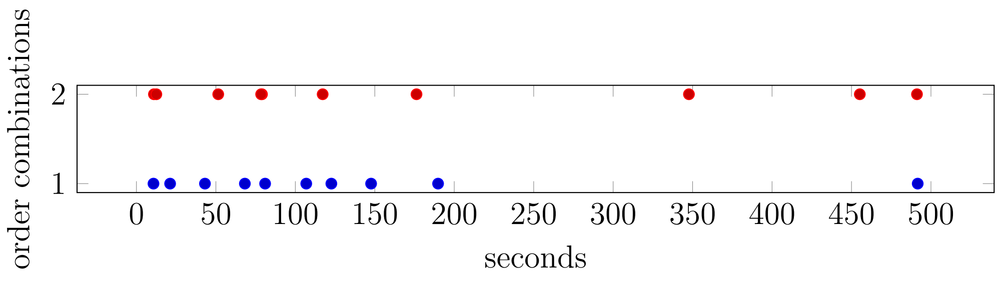


Figure 13: 1:BYYYG+BGG+ROG, 2:SBG+SGG+SOG, all with 900 sec deadline and all satisfiable

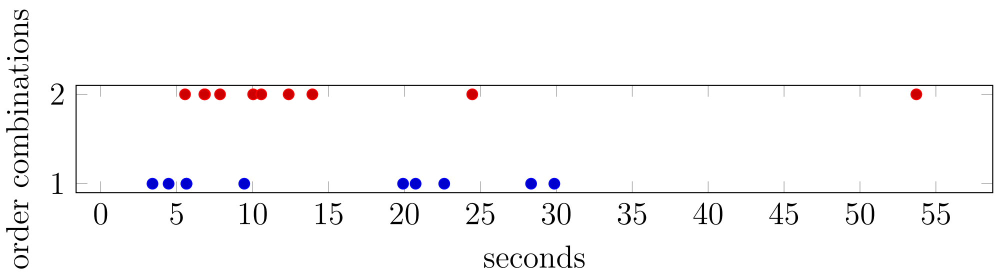


Figure 14: 1:BOOOG, 2:SOG+RG+BBB, all with 900 sec deadline and all satisfiable

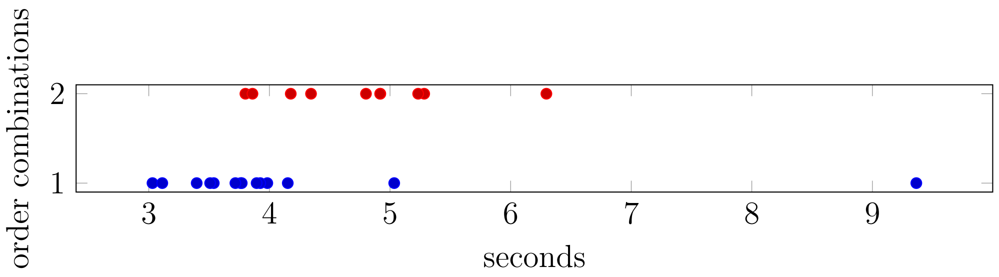


Figure 15: 1:BYYYG+RB, 2:SG+SBG+SGG, all with 900 sec deadline and all satisfiable

Since shortening the deadline decreases the amount of possible action sequences, one can expect that in this case also the solving time increases. Figure 16 shows a significant gap in the solving times between a solution with a tight deadline and the proof of the unsatisfiability of an instance. This leads to the hypothesis, that if a solution exists it will be found relatively fast compared to the time needed to prove that no action sequence exists which serves all orders. It might be beneficial to assume the unsatisfiability of an instance after a certain time and try another one with lower product complexity or less orders.

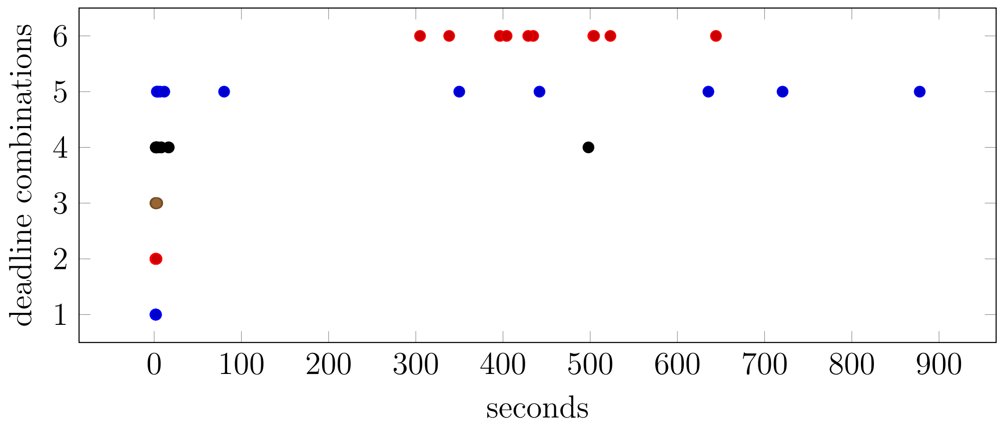


Figure 16: BGGGG with varying deadlines, 1: 900 sec, 2: 800 sec, 3: 700 sec, 4: 600 sec and one UNSAT, 6: 550 sec and 6 UNSAT 6: 500 sec and all UNSAT, all UNSAT after the 50 sec mark

Figure 17 shows a significant increase of the solving time when searching for a model which minimizes the occupation time of the delivery station in the last world state instead of searching an arbitrary model.

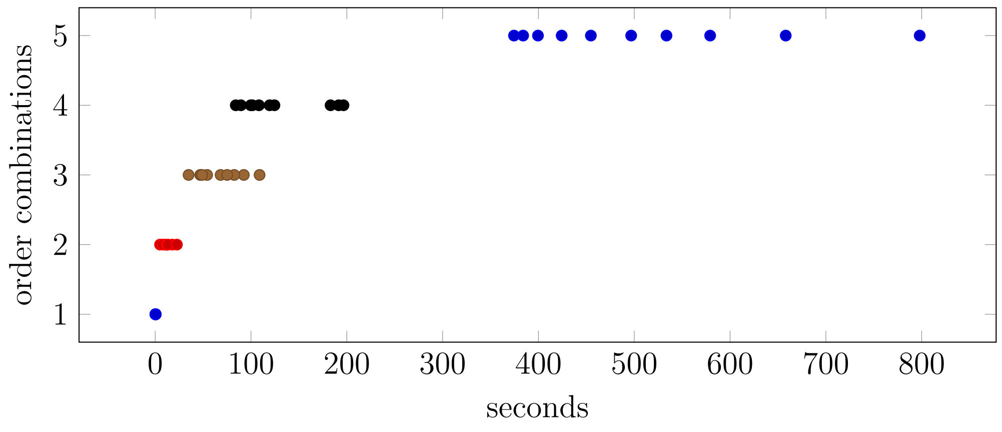


Figure 17: 1:BG, 2:BBG, 3:BGG, 4:BBBG, 5:BOG, using the optimizer to minimize the delivery station occupation time, all satisfiable

5 Conclusion

This thesis addresses the planning problem for the RoboCup Logistics League game by means of a SMT-solver. Therefore, first, a brief overview about the used technologies of SMT-solving has been given, followed by a description of the RoboCup Logistics League. Then, the problem has been abstracted and a general approach to solve planning problems by means of an SMT-solver has been presented. Subsequently, the specific formula creation for the RoboCup Logistics League has been introduced, including two different encoding variants, one using a plain SMT-solver, the other an *optimizing* SMT-solver. With regard to the above described measurement results, it can be summarized that the approach using an optimizing SMT-solver resulted in considerably larger solving times than the approach using a plain SMT-solver. Therefore, it can be concluded that the approach using an optimizing SMT-solver does not seem feasible with complex and combined orders in its current form, whereas the approach using the plain SMT-solver shows promising first results. Nevertheless, as will be outlined in the subsequent section more detailed, further research is needed to prove the realizability of the presented approaches.

5.1 Future Research

One of the most relevant questions for future research is, whether the two approaches are feasible in a game of the RoboCup Logistics League. For example, it needs to be examined whether the assumption of constant travel times between stations is a sufficiently good description. In addition, the solving speeds of the formulas have to be further increased. One possibility is to add constraints to provide implicit encoded information explicitly to the solver, allowing the proving the unsatisfiability of instances faster. Another possibility would be an alternation of the encoding, which may result in a reduced variable count and action sequence length, thus increasing the solving speed. Furthermore, additional experiments are needed in order to be able to predict the solving time under certain conditions more precisely and to proof the reliability of above made conclusions.

Reduce Variable Count and Action Sequence Length Under the assumption that not two stations of the same type can mount the same color, redundant information is encoded if all steps of the order progression are tracked in the way Chapter 4.3.2 describes. The information about a station holding a workpiece does not need to be encoded explicitly, as it can be derived from the order status. While the information which robot performed the needed actions for the progression step can be read from the time variables.

Modifying the segmentation of the order progression in a way that the corresponding actions leave the workpiece always in the output of a station, the information whether a robot holds a workpiece is obsolete. That means, the action of collecting a base is included in the order status "first ring mounted" and "additional base fed". This approach does not restrict the set of possible action sequences since a robot collects a base with the aim of mounting a ring or with the aim of feeding it into a ring station without dropping it in between. The action of setting up a ring station for a particular color can also be included in the status "ring mounted" or if additional bases are needed in the status "first additional base fed". These alterations are pictured in Figure 18.

The transitions between the order states can be encoded as known from Chapter 3 and the time encoding can be inherited from Chapter 4.1.1. All information needed to update the time properties in an adequate way is available when transitioning to a new order state. The location of the product in the previous step can be derived since the color of the mounted component is known and the station mounting this color is unique.

An additional status "transparent base removed" has to be introduced to indicate whether the transparent base after feeding the cap is removed. Transparent bases can be fed into ring stations if the status "cap fed" is set, in case this possibility is chosen, the time properties have to be set properly and the status has to be set to "transparent base removed". Otherwise the status can be altered by dropping the transparent base by a robot.

This encoding reduces the needed steps to serve an order from 6-27 steps to 3-12 steps compared to the old encoding. Additionally, the variable count to represent a world state is reduced by at least 30 for the workpiece representation for each machine.

A drawback is that the scheduling can only start with a world state in which all workpieces are placed on machines, this makes the use of another planner necessary to restore this state before.

Speeding up the Solving Process by Adding Additional Constraints
Constraints to check in each step whether the corresponding order can still be delivered under optimal circumstances could be added. In each step the sum of the travel times to reach all other in the future needed stations, the corresponding processing times, and the already elapsed time must not exceed the deadline.

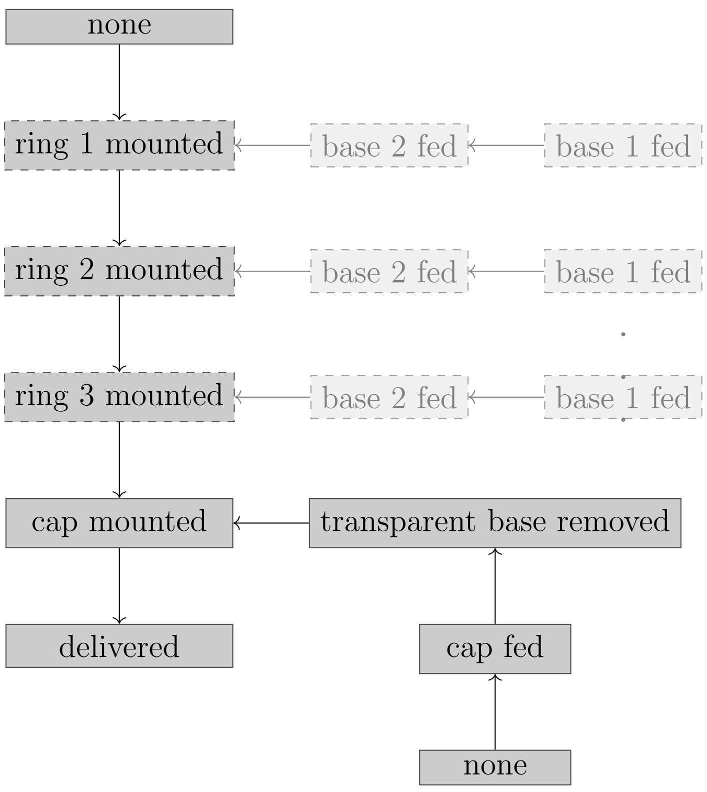


Figure 18: Modified order progression

References

[ÁK16] Erika Ábrahám and Gereon Kremer. Satisfiability checking: Theory and applications. In *Proc. of SEFM'16*, pages 9–23, 2016.

[ÁLCS10] E. Ábrahám, U. Loup, F. Corzilius, and T. Sturm. A lazy smt-solver for a non-linear subset of real algebra. `http://www-i2.informatik.rwth-aachen.de/pub/index.php?type=download&pub_id=596`, 2010.

[BFT16] Clark Barrett, Pascal Fontaine, and Cesare Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). `www.SMT-LIB.org`, 2016.

[BPF15] Nikolaj Bjørner, Anh-Dung Phan, and Lars Fleckenstein. νz - An optimizing SMT solver. In *Proc. of TACAS'15*, pages 194–199, 2015.

[BSST09] Clark W. Barrett, Roberto Sebastiani, Sanjit A. Seshia, and Cesare Tinelli. Satisfiability Modulo Theories. In *Handbook of Satisfiability*, pages 825–885. 2009.

[BST10] Clark Barrett, Aaron Stump, and Cesare Tinelli. The SMT-LIB Standard: Version 2.0. In A. Gupta and D. Kroening, editors, *Proceedings of the 8th International Workshop on Satisfiability Modulo Theories (Edinburgh, UK)*, 2010.

[CBRZ01] E. Clarke, A. Biere, R. Raimi, and Y. Zhu. Bounded model checking using satisfiability solving. *Form. Methods Syst. Des.*, 19(1):7–34, July 2001.

[CKJ$^+$15] Florian Corzilius, Gereon Kremer, Sebastian Junges, Stefan Schupp, and Erika Ábrahám. : An open source C++ toolbox for strategic and parallel SMT solving. In *Proc. of SAT'15*, pages 360–368, 2015.

[DK08] R.E. Bryant D. Kroening, O. Strichman. *Decision Procedures: An Algorithmic Point of View.* Texts in Theoretical Computer Science. An EATCS Series. Springer, 1 edition, 2008.

[dMB08] Leonardo de Moura and Nikolaj Bjørner. *Z3: An Efficient SMT Solver*, pages 337–340. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[DNK$^+$15] C. Deppe, T. Niemueller, U. Karras, A. Rohr, T. Neumann, and W. Uemura. *RoboCup Logistics League Rules and Regulations 2016*, December 2015. `http://www.robocup-logistics.org/rules/rulebook2016.pdf`.

[FM09] John Franco and John Martin. *Handbook of Satisfiability*, chapter A history of satisfiability, pages 3–74. IOS Press, 2009.

[HNCL16] Till Hofmann, Tim Niemueller, Jens Claßen, and Gerhard Lakemeyer. Continual planning in Golog. In *Proc. of AAAI'16*, pages 3346–3353, 2016.

[KWJ13] H. Kagermann, W. Wahlster, and J.Helbig. *Recommendations for implementing the strategic initiative INDUSTRIE 4.0. Final report*. Platform Industrie 4.0, 2013. `http://www.acatech.de/fileadmin/user_upload/Baumstruktur_nach_Website/Acatech/root/de/Material_fuer_Sonderseiten/Industrie_4.0/Final_report__Industrie_4.0_accessible.pdf`.

[LÁN$^+$ar] Francesco Leofante, Erika Ábrahám, Tim Niemueller, Gerhard Lakemeyer, and Armando Tacchella. On the synthesis of guaranteed-quality plans for robot fleets in logistics scenarios via optimization modulo theory. In *Proc. of IRI'17*, to appear.

[NER$^+$14] T. Niemueller, D. Ewert, S. Reuter, A. Ferrein, S. Jeschke, and G. Lakemeyer. *RoboCup Logistics League Sponsored by Festo: A Competitive Factory Automation Testbed*, pages 336–347. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.

[NKVT16] T. Niemueller, E. Karpas, T. Vaquero, and E. Timmons. Planning competition for logistics robots in simulation. https://www.fawkesrobotics.org/media/publications/logrobcomp-icaps2016.pdf, 2016.

[NLF13] Tim Niemueller, Gerhard Lakemeyer, and Alexander Ferrein. Incremental task-level reasoning in a competitive factory automation scenario. In *Proc. of AAAI'13 Spring Symposium*, 2013.

[NLF15] Tim Niemueller, Gerhard Lakemeyer, and Alexander Ferrein. The RoboCup Logistics League as a benchmark for planning in robotics. In *Proc. of PlanRob@ICAPS'15*, 2015.

[NLLÁar] Tim Niemueller, Gerhard Lakemeyer, Francesco Leofante, and Erika Ábrahám. Towards CLIPS-based task execution and monitoring with SMT-based decision optimization. In *Proc. of PlanRob@ICAPS'17*, to appear.

[NO06] Robert Nieuwenhuis and Albert Oliveras. On SAT modulo theories and optimization problems. In *Proc. of SAT'06*, pages 156–169, 2006.

[Rin15] J. Rintanen. Discretization of temporal models with application to planning with smt. In *AAAI Conference on Artificial Intelligence*, 2015.

[ST15a] Roberto Sebastiani and Silvia Tomasi. Optimization modulo theories with linear rational costs. *ACM Trans. Comput. Log.*, 16(2):12:1–12:43, 2015.

[ST15b] Roberto Sebastiani and Patrick Trentin. Pushing the envelope of optimization modulo theories with linear-arithmetic cost functions. In *Proc. of TACAS'15*, pages 335–349, 2015.