CHALMERS
UNIVERSITY OF TECHNOLOGY

Sequence Planning and Optimization of Production Systems

Implementation of a tool to Automatically Generate Optimal Sequences of Instructions for Production Systems

Master's thesis in Systems, Control and Mechatronics

JOHAN ASKLUND and NAEMI JÖNSSON

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2017

Master's thesis 2017:EX097

Sequence Planning and Optimization of Production Systems

Implementation of a tool to Automatically Generate Optimal Sequences of Instructions for Production Systems

JOHAN ASKLUND
NAEMI JÖNSSON

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Division of Systems and Control
Automation
Chalmers University of Technology
Gothenburg, Sweden 2017

Sequence Planning and Optimization of Production Systems
Implementation of a tool to Automatically Generate the Optimal Sequence of Instructions for Production Systems

Supervisor: Henrik Carlsson, Volvo Cars Corportion
Supervisor: Martin Dahl, Chalmers Univeristy of Technology
Examiner: Petter Falkman, Chalmers University of Technology

Master's Thesis 2017:EX097
Department of Electrical Engineering
Division of Systems and Control
Automation
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Illustration showing part of an operation sequence for one robot, generated by Sequence Planner.

Typeset in LaTeX
Gothenburg, Sweden 2017

Abstract

This thesis mainly concerns the further development of Sequence Planner (SP), a tool used for verification and optimization of operation sequences. The work was conducted on a virtual robotic production cell (at Volvo Cars) modeled in the virtual commissioning software Process Simulate (PS). A way to handle more complex robot sequences and other resources, such as fixtures was also developed in this project.

Sequence Planner is now capable of importing and modeling the robots as well as some fixtures from the virtual PS model. The SP model can be manually modified and added to, whereafter it can be verified by means of supervisory control theory, ensuring a non-blocking and deadlock free system. Using constraint programming, a set of feasible operation sequences can be produced and sorted according to their total execution time. The sequences can then be sent back to the virtual commissioning tool for further testing and verification.

Keywords: Sequence Planner, Robot Cell, Volvo Cars, Chalmers, Industrial Automation, Industry 4.0, Optimization, Constraint programming, Supremica

Acknowledgements

We give special thanks to our supervisor Henrik Carlsson and manager Hans Hörfelt at Volvo Car Corporation (VCC) for the time and effort put into the project. Additionally we thank Robert Brännare and the PLC department for explaning the PLC program and coding conventions to us.

A word of gratitude to Alexandru Dinca from Summ Systems is also warranted for giving us an introductory course in Process Simulate and TIA portal.

Lastly we thank Martin Dahl and Petter Falkman, our academical supervisor and examiner from Chalmers University of Technology for their support in the project and with Sequence Planner.

Johan Asklund and Naemi Jönsson, Gothenburg, September 2017

Contents

List of Figures

1
Introduction

Today automation is a hot topic all over the world, where both the number and complexity of tasks performed by machines are constantly increasing. Consequently the demand for more efficient and complex software tools is rising.

The goal of this thesis is to further develop a tool that automatically generates time optimal sequences of instructions, for robots and other moving components in a production system. The tool will be developed for a production cell at *Volvo Cars Corporation (VCC)*.

1.1 Background

Much of the work in modeling, testing, programming and setting up production systems is currently performed manually. There exists modeling and programming tools that can simplify the job, but there is room for improvement. A common standard and working method for these tools, as well as more automation and potentiation is desirable. A higher level of automation would greatly benefit both the workers and the production efficiency, cutting down on repetitive tasks, minimizing mistakes, and creating more optimized operational sequences for the production cells.

A step in this direction is to use a tool that automatically generates optimal operational sequences for every autonomous component in a cell. These tools are rare, but since the demand is increasing, companies and universities have started to develop them. One such tool for sequence planning is called *Sequence Planner (SP)* [1] and is currently being developed by Chalmers University of Technology's research group of automation.

Our thesis is part of a larger project called *VirtCom*, which is a collaboration between Chalmers and other companies, about the subject *virtual commissioning (VC)* (VC is described in Section 2.1). There are two other VirtCom theses that will be pursued at VCC during 2017, namely "Automatic Collision Avoidance for Robots" and "Virtual Commissioning", both closely related to our thesis.

1.2 Objectives

The main part of our project was to further develop the tool SP for a virtual production cell at Volvo Cars. The production cell in question was formed around a core of robots which transported material between stations and performed welding operations, hence it is called a robot cell. For the first part of the project, only the robots were considered for the modeling and optimization. Then, other machines were gradually added. Volvo Cars provided a virtual model of the robot cell made in the software *Process Simulate (PS)*.

One of the challenges was to make sure that SP could easily import data from PS and fashion a model of the cell, for subsequent optimization.

Another important part in our thesis was to integrate SP into Volvo Cars work methodology to help them automatically generate optimal sequences for their robot cells. As mentioned above, Volvo already have existing models of their production cells. Therefore we wanted to use as much information from the model assigned to us as possible. If changes to the model or additional information was required, then it should be made in a way that complies as much as possible with Volvo methodology, simplifying future work.

The research question for the project was consequently: *How can Sequence Planner be developed to optimize robot cells at Volvo Cars? And at the same time keep the changes in Volvo Cars existing methodology minimal.*

The objective can be divided into the areas listed below.

1.2.1 Key areas

1. Implementation of the model in Sequence Planner
   (a) Import data using an already existing plugin for Process Simulate, the plugin can be modified if necessary.
   (b) Make any necessary adjustments to Sequence Planner or the model.
2. Add more components and information to the model:
   (a) Augment the model.
   (b) Ascertain that Sequence Planner still works as intended, and can perform synthesis and optimization.
3. Modification of Sequence Planner and additional functionality:
   (a) Propose changes to Sequence Planner.
   (b) Decide which to implement.
   (c) Implement changes.
   (d) Test and verify them.

1.3 Robot cell

For testing and verification purposes VCC provided a virtual model of the robot cell *17-36-070 Load Wheelhouse* from their factory in Torslanda. The cell is part of a larger production system called cluster 90, which produces the car models S90, V90 and XC90. In this particular cell, the wheelhouses are attached to the rear floor for each respective car model.

The robot cell, displayed in Figure 1.1, consists of seven robots, two turn tables, seven fixtures (where six of them are attached to the turntables) and a couple of buffers, welding-, and gluing stations. Figure 1.2 indicates the general operation order of the cell with arrows, for a more detailed view with component naming see Figure A.3.

The cell is controlled by a programmable logic controller (PLC) in conjunction with the robot programs. The PLC handles the overall interaction between different resources of the cell and determines which car model is up for production. To prevent collisions in the cell, zones have been established in the PLC, where only one robot can enter at a time, see colored parts of Figure 1.2.

Depending on the incoming parts of the cell, the PLC will decide upon which model to produce, which robot schedules will be active and what position the turntables should be in.

For a normal cycle of the cell, the rear floor enters through the conveyor, where a robot is waiting to spot weld it and place it in the middle fixture. At the same time the left and right wheelhouses are prepared to be attached to the floor by adding glue to the components. The glue may have to pass a quality check before it is placed onto the turn table. When both wheelhouses and the rear floor are in place they are welded together and the assembled part is then moved to the next station.

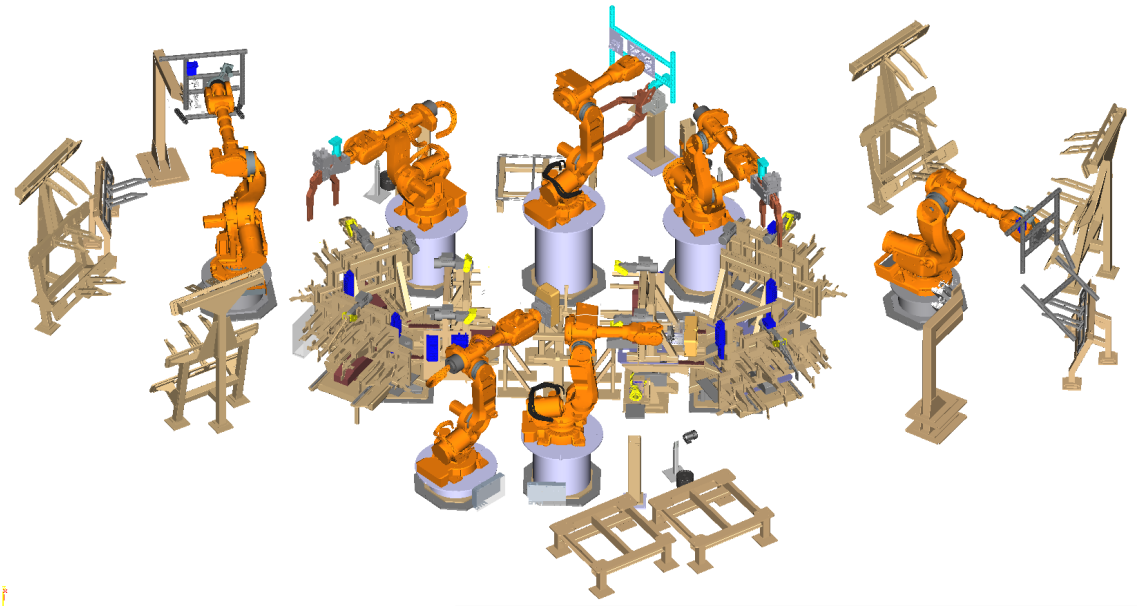

Figure 1.1: The PS model.

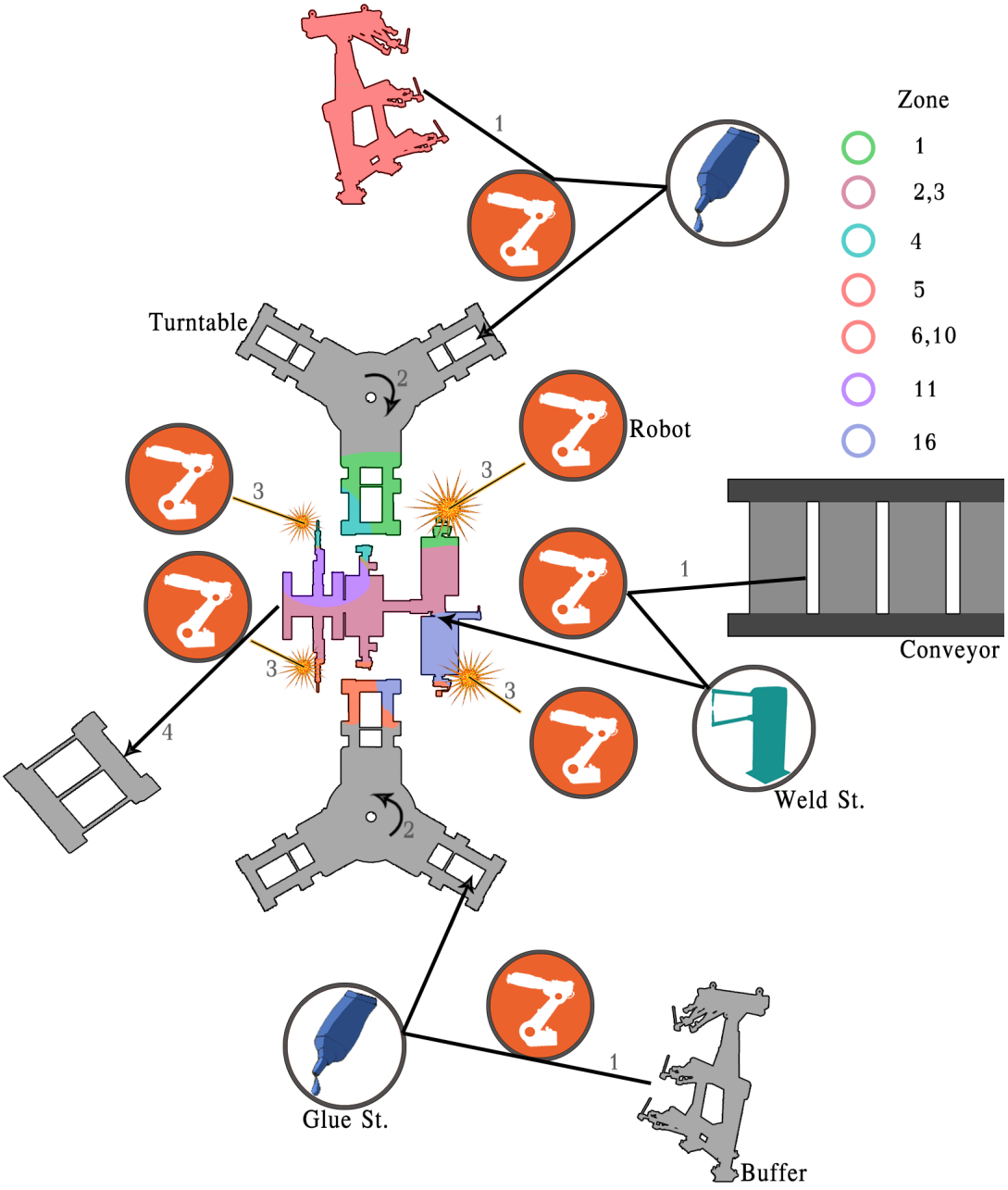

Figure 1.2: A simplified overview of the robot cell. The large round symbols denote either robots, gluing stations or a weld station. To the right of the cell is a conveyor. On top and bottom are buffers and in the middle is a fixture with turntables on both sides. The lined arrows with numbers show the production flow, where parts enter from the buffers and the conveyor. In the buffers, wheelhouses for different car models are collected. From the conveyor, rear car floors emerge. The zones listed with annuli to the right are represented in the cell with the same colors.

1.4 Delimitations

This thesis work was conducted at Chalmers University of technology and Volvo cars. It is part of a larger project where one group was working with mapping the virtual model to the PLC, another group with automatic zone generation for collision avoidance, and our project, which mainly concerned the further development of Sequence Planner. We wanted to integrate these theses, but due to limited time and compatibility issues we did not.

Our goal was to incorporate Sequence Planner with Volvo's methodology, and to further automate the process of generating robot sequences. To that end, we used Volvo's current software and programs, including Process Simulate and Totally integrated Automation Portal (TIA-portal), further explained in Section 3.

To communicate between PS and SP we made use of and elaborated on a plugin for PS, which has been developed by our supervisor Martin Dahl.

Since the programs subject to change are written mainly in Scala, Angular JS, HTML and *C#*, we used these languages for our implementations and modifications.

SP as a whole was being restructured during our project, where the graphical user interfaces (GUIs) were being rewritten, using Scala JS. Therefore we did not focus to much on the old GUI.

We ended up using the optimization algorithm that was already implemented in SP at the start of the project, described in Section 3.1.4. We had considered improving upon its implementation and usage, or trying other algorithms, but settled for the existing implement due to limited time.

With this thesis work we wanted to find good operation sequences. Further optimization such as optimizing the robot path or minimizing mutual zones were not to be investigated.

Volvo provided us with a virtual model of a robot cell and our tests were exclusively performed on that cell. Although trying to make the method as general as possible, we cannot ensure that it works for all other production cells at Volvo.

2
Theory

For a comprehensive understanding of the thesis work, an overview of the basic concepts is presented here. The main part of the project was to develop Sequence Planner for verification and optimization of operation sequences.

The optimizer uses constraint programming (CP) [2] with constraints given by execution order and mutually exclusive operations.

The verification is done in two ways. One is Virtual Commissioning (VC) which is the same method VCC is using today, graphically simulating a model of the production system. VC can however only guarantee correct behaviour for specific tested scenarios, as production systems grow larger and the number of possible states increase, this becomes a slow and tedious process.

Thus another method is also desired, namely formal verification using supervisory control theory (SCT) [3], creating a mathematical model of the system, which can be used to prove correct behaviour.

2.1 Emulation and Virtual Commissioning

Simulation is a common way to test and verify new solutions before implementing them in the real system. However, even though simulation gives a good overview of the system, it is a simplified model of reality [4], making it hard to find problems that only affect small parts of the system. Therefore, emulation has become popular recently. Emulation is similar to simulation but instead of giving a good overview it focuses on one particular process and imitates the real system as closely as possible [4]. The most common way to use emulation in manufacturing is emulation of Logical Validation, also known as Virtual Commissioning (VC). The main purpose of VC is to identify and eliminate every possible failure, before implementation in the real system [5]. Other benefits of VC is that it gives an accurate picture of the system and makes late changes to the logic possible and affordable.

Since VC tries to emulate the real system as accurately as possible, certain information is required when realizing the system [6] which is described below.

Model
: A realistic 3D model of the system that includes geometries, kinematics, electronics and controller programs for all moving components.

Production cell layout
: An accurate representation of the production cell's layout, including exact placement of all components and equipment.

Flow of material
Clearly stated sequences for the production system operations.

Programming
The code for the system (e.g. PLC and robot code) written in the same way as in the real system. Many modern VC tools run the same code for emulations as is used in the real systems.

Input/outputs
Detail mapping and definition of the input/outputs for the systems.

Extra functionality
Clearly defined extra functionality for the system that should be included in the VC, such as safety systems etc.

IT infrastructure
Specification of the IT structure, such as software drivers and communication protocols.

2.2 Formal Verification

VC gives a graphical verification of the production system, but without extensive testing there is no guarantee that it works for all possible cases [7]. Formal Verification (FV) in comparison, uses a mathematical approach to prove correctness of algorithms. The first step in FV is to create a mathematical model of the system. The model in our case is discrete, based on states and event based transitions. Thus it is possible to analyze all unwanted states and determine if there is any possible way of reaching them. When the unwanted states are identified a supervisor can be synthesized to stop the system from entering the forbidden states. For example, if two robots cannot occupy the same space at the same time, then a shared zone surrounding that space can be created, which only one robot can enter at a time, solving the problem. This approach of FV is called supervisory control theory (SCT) and is implemented in SP using the tool Supremica [8].

2.3 Constraint Programming

Constraint Programming (CP) [2] tries to solve problems containing multiple variables subject to constraints, by changing values of the variables to fulfill the constraints. This way the variables can take on many values, creating several feasible solutions. To find an optimal solution the feasible solutions are compared to some criteria, in our case minimizing execution time.

3
Software

The main software used in this project is SP, which consists of several sub services, including the optimizer and FV tools. The service we primarily will be focusing on is called Volvo robot scheduling tool, developed by Martin Dahl.

The virtual commissioning tools in use at Volvo that we will be touching on are PS for modeling and verification of the production cells, and TIA-portal for developing and simulating the PLC code.

3.1 Sequence Planner

Sequence Planner [9] is a veritable toolbox for process modeling, visualization, optimization [10], observation and control [11], developed by Chalmers research group of automation.

3.1.1 Architecture

Built on a microservice architecture [12] using messages over a shared bus to communicate between services, Sequence Planner's services can be set up and modified independently from each other using different languages. Because of this modularity it is easy to adjust and add features. The program itself is freely available from the Chalmers researchers at GitHub [13].

3.1.2 Modeling and visualization

One of SPs features is modeling of processes and their visualization. The models are built using sequences of operations (SOPs) [14], consisting of extended finite automatas (EFAs) [15] in the SP-language (SPL), where sequences are based on relations of when different operations are allowed to execute. These sequences can be visually inspected from several perspectives, e.g. as a whole or from the individual operations, to the resources performing them [1, 16]. In Figure 3.1 is an example of how an SOP for two robot sequences can look in SP, the zone specification SOP in the picture will only allow one of the zone operation sequences to execute at a time.

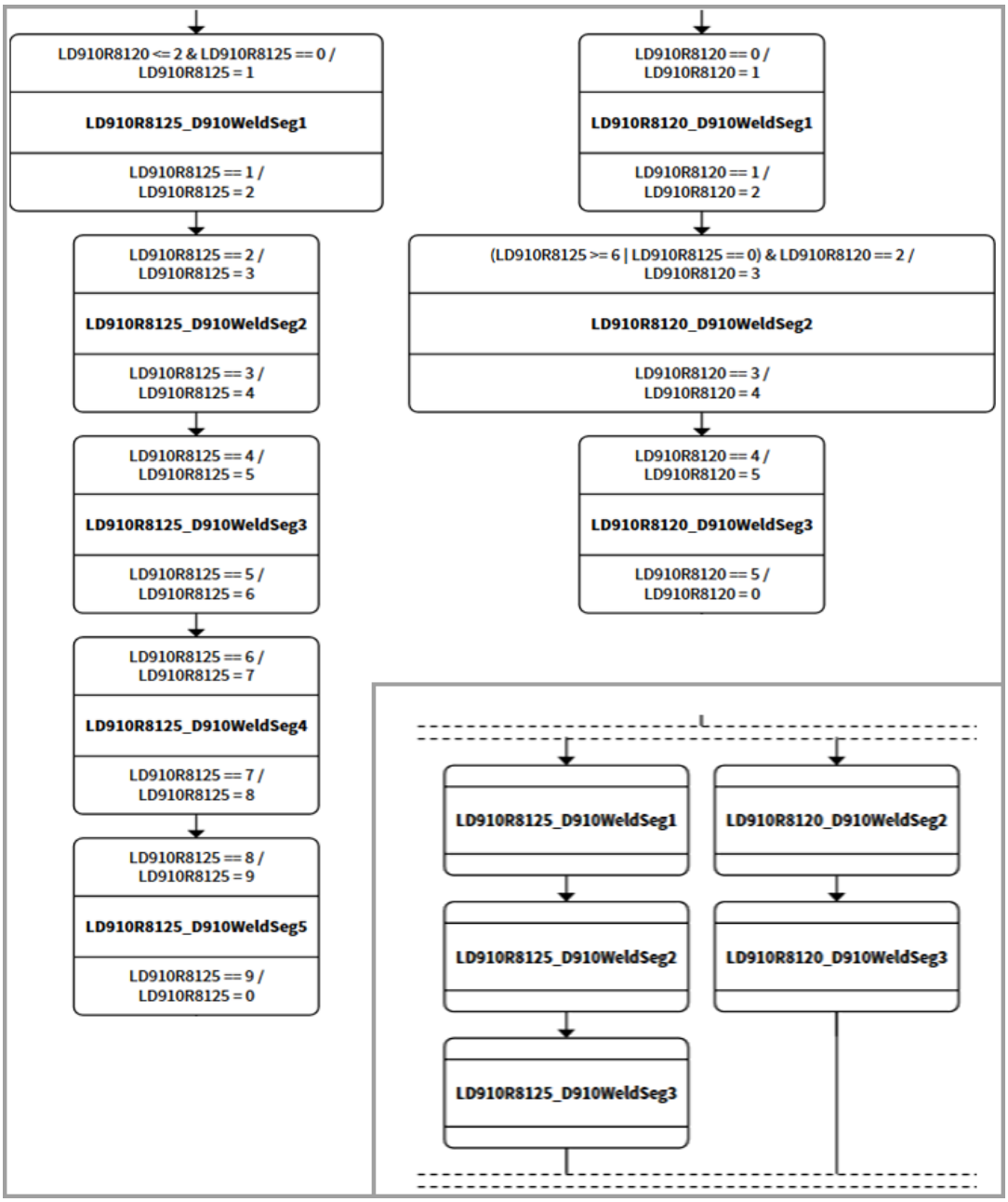

Figure 3.1: Example of two simple robot sequences from SP at the start of the project. On top is an SOP for the two robots with synthesized guards from Supremica. In the top of the boxes are the guards with preconditions that should be fulfilled before the second line (i.e. the action) below the / can start. When the action/ operation has been performed, the post condition in the lower part of the box will be fulfilled and so the next guards and operations can be evaluated and executed. Lowermost is an SOP zone specification, containing mutually exclusive operation sequences for the robots. The zone specification adds additional guards to the operations, also shown in the picture. The guards added from the zone specification will ensure that only one of the operation sequences contained in the zone can execute at a time. So for example if robot 8120 has started to execute its operation weldSeg2, then robot 8125 is not allowed to execute any of its WeldSeg(1-3) operations untill robot 8120 is done with its operation WeldSeg3.

3.1.3 Formal verification

The sequences can be verified using tools such as Supremica [8], where Supervisory Control Theory schemes [3, 17] are applied to the SP model to create a minimally restrictive, controllable and non-blocking supervisor. In Figure 3.1, guards generated from the SOP sequences and zone SOP sequences are shown with the operations.

3.1.3.1 BDD verification

To test that the supervisor is working as expected a binary decision diagram (BDD) verifier [18] - where the user can select different states of the operations for each resource to see if they can execute at the same time - also exists in SP, see Figure 3.2.

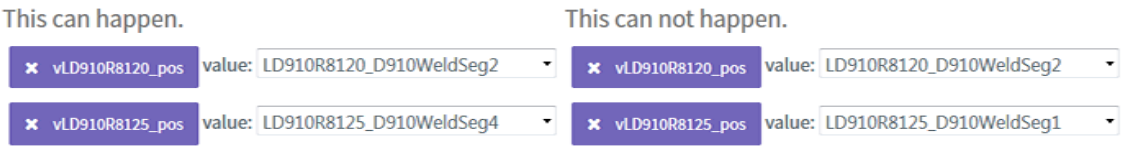

Figure 3.2: Using the BDD verifier on the synthesized model in figure 3.1. To the right we tried seeing if the two robots could execute operations in the same zone, at the same time, this does not work as expected. To the left only one of the robots operations is within the zone, so that is alright.

3.1.4 Sequence optimization

The SOP model is optimized with constraint programming (CP) [19,20] in the same step as the synthesis, where constraints concerning sequences and common zones are formed for all resources. In this approach the SP model first has to be transformed to a CP model [9,19] before a solver such as OscaR [21] - which is used in SP at the moment - can take a crack at the problem.

From the order in which the operations are modeled in the SOP, constraints will be created to ensure that the operation order for each resource is maintained. Every operation has an associated execution time which is used to determine when it can begin execution. An operation can only start executing if it is the first operation in the sequence or after another operation is done. The zone SOPs makes sure constraints are formed so that only one sequence in each zone can start at a time. For each zone sequence, additional constraints are added to reduce complexity of the problem, saying that all operations in that sequence have to execute immediately after each other.

The solver will produce a set of feasible solutions to the problem (within a specified time limit on how long the solver is allowed to run), with a new SOP and gantt chart for each one, see Figure 3.3. The gantt charts will showcase the different solution sequences, and the SOPs will be extended with additional conditions to make the operations follow those sequences.

Some sequences could be more or less time optimal, for instance it could be that in one solution all operations that can execute in parallel do so, giving the fastest

sequence. While in another solution no operations execute in parallel, giving a slower sequence.

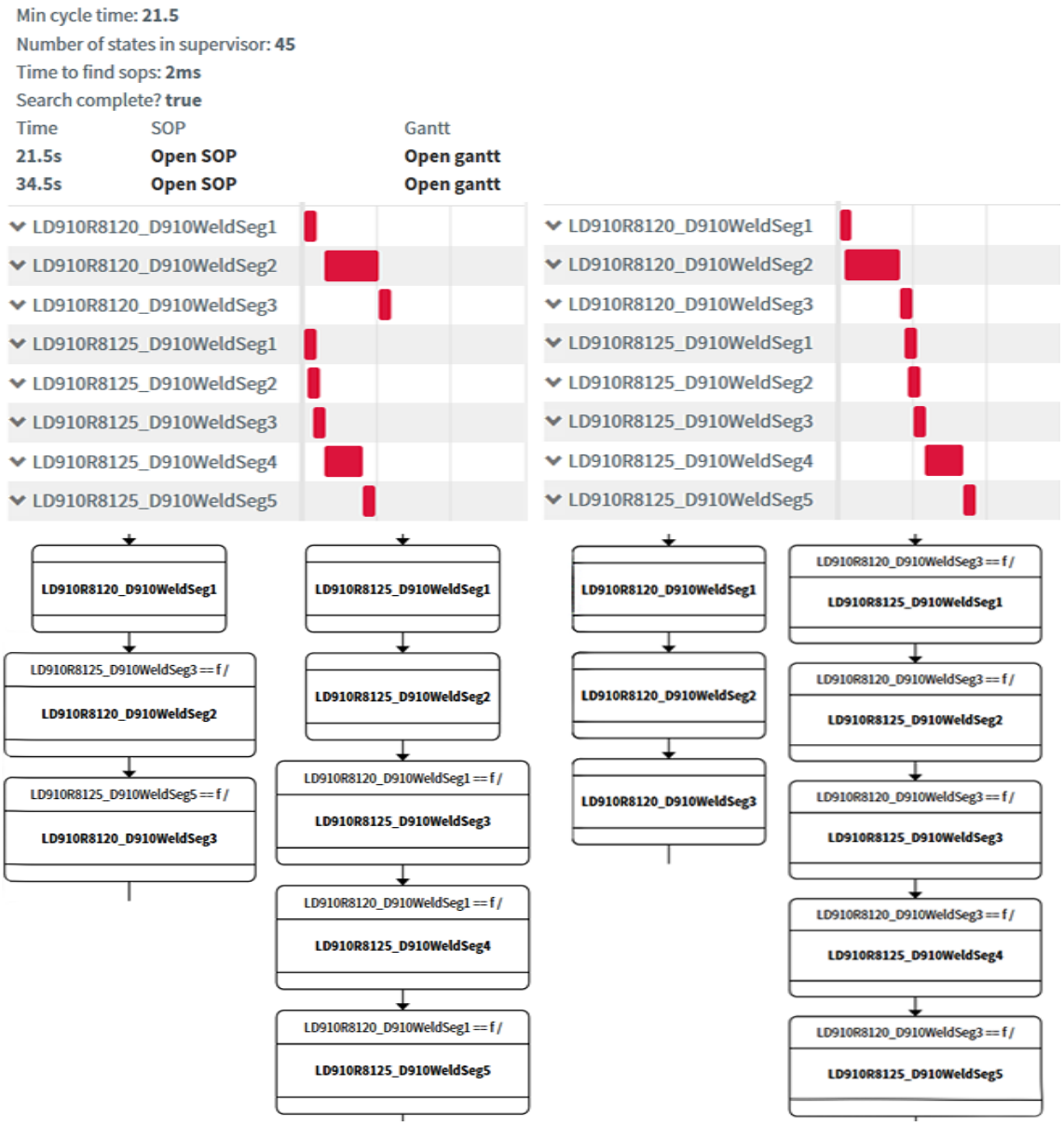

Figure 3.3: Two solutions to a CP sequence optimization problem, where additional conditions have been added to the SOPs, so that the operations will later execute in the order of the gantt charts, when exported to PS.

3.2 Process Simulate

Process Simulate is a program developed by Siemens Tecnomatix [22], which is a tool Volvo uses for Virtual Commissioning.

In Process Simulate a virtual manufacturing model is constructed using resources that constitutes robots, machines, fixtures, tools and workers. Individual resource 3D models of the Process Simulate-model are usually created in third party computer-aided design (CAD) software. For easy transition from the original CAD-models a multi-CAD standard JT [23] format has been developed which provides a lightweight 3D model with associated product data. The kinematic data for the 3D components is however modeled in Process Simulate. If instead the JT format was combined with STEP 242 [23] the kinematic data could be modeled in the original CAD software, possibly making the overall process easier.

In addition to the resources, the model requires logical operations to perform any virtual commissioning. However the logic is implemented in different ways depending on the resource. Compared to clamps and turn tables, robots have a lot more freedom in terms of movements and possible actions, thus PS has a special method of adding logic for robots.

It is possible to create robot programs in PS [24] and to import existing robot programs, converting them to PS robot operations. The later alternative is what VCT have done when building their virtual commissioning model.

Adding logic for PLC controlled components is more complicated. PS does not have a built in function to emulate the PLC signals. Fortunately, it is possible to emulate PLC signals using so called logic blocks [24] and connecting them with a real PLC or external software that can emulate PLC signals.

After setup is complete, PS provides analysis tools to detect possible collisions, shortest time to complete a production cycle and optimization of the robot paths. Despite these advanced features, there is not yet a good way of ensuring intended system behaviour in a formal fashion using only this type of tool.

3.3 Process Simulate plugin

It is possible to create your own plugins for PS. For example Nina Sundström [25] developed a plugin that enables the user to select resources and associated operational information from the PS environment, exporting it to XML files that Supremica can in turn read, synthesize and hand to Sequence Planner for visualization.

Martin Dahl developed a plugin called Tecnomatix plugin, creating an interface where desired information such as operation and robot program data can be sent directly to SP from PS. SP can from the PS robot programs create a model of the system, from which it can verify and optimize sequences that can be visualized and sent back to PS through the plugin for execution and testing. This plugin is further explained in Section 4.1, where our use and development of it is presented.

3.4 Totally Integrated Automation Portal

Totally integrated Automation Portal (TIA-portal) is a software for developing and controlling PLC systems. It has the capability of setting up PLC systems, both for real- and virtual PLCs, and to write logic control programs [26]. Since both TIA-portal and PS is software provided by Siemens, they are compatible. Meaning that the PLC in TIA can be used to test and verify the virtual model in PS, before uploading to the real system.

4

Developing SP's capabilities

This chapter describes the main part of the project, from importing data to SP, to how SP has been modified to handle more complex robot cells.

4.1 Importing information from Process Simulate to Sequence Planner

One of the first steps in the project was to send all desired data from PS to SP. The Tecnomatix plugin of Section 3.3 written in *C#* provided the majority of the desired information. However it could not handle alternative branches, where a robot could have several options when executing the next operation. The plugin has been modified to loop over the branches and extract the operations within recursively, as Listing 4.1 describes. The plugin is used as in Figure 4.1, where it is possible to select what to import, from individual hierarchies to all operations.

Listing 4.1: PS plugin, importing hierarchies pseudo code, where the input ID should at first be the selected hierarchy root in SP.

```
function GetPSOperationHierarchy(ID)

  operation = Get PS operation which corresponds to the ID
  init array of children
  if(operation has descendants)
    foreach(descendant in descendants)
      add GetPSOpHierarchy(descendant ID) to array of children

  return (operation with children)
```

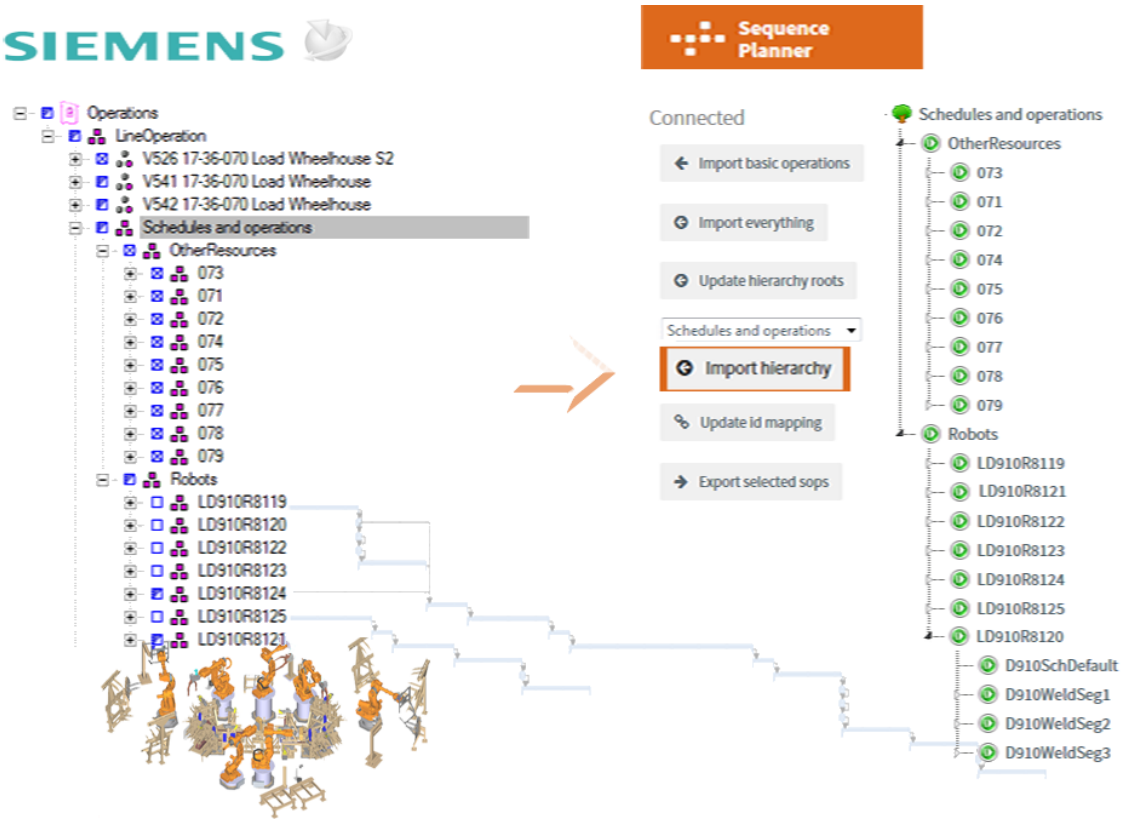

Figure 4.1: Importing the PS operation hierarchy to SP.

4.2 Splitting the SP program

At the start of the project, SP could after importing robot schedules from PS generate and optimize the SOPs in one step. We however wanted to add the ability to inspect and modify the SOPs after generation, before synthesis and optimization. This is a necessary step if we for example would like to manually add a model of the PLC at a later stage. So the existing program had to be divided in two parts. One for generating SOPs and one for running the synthesis and optimization.

4.3 Generating SOPs

The creation of SOPs was also divided in two parts, one where the robot schedules are processed and one for other resources such as turntables and fixtures. For each resource, zone SOPs and an SOP for the operation flow are created. When all SOPs for the resources are done, they are combined to common zone SOPs and one SOP for the actual operation sequences according to Listing 4.2. The SOPs and operations are then used to create a new hierarchy in the hierarchy tree of SP as in Figure 4.2.

Listing 4.2: SOP generation - pseudo code

```
SOPs =resources.map {resource

    if( resource == 'Robot')
        AddRobot(resource.schedule)
    else
        AddOtherResource(resource)
    }
Combine the SOPs
Return a new hierarchy where all SOPs and operations are contained
```

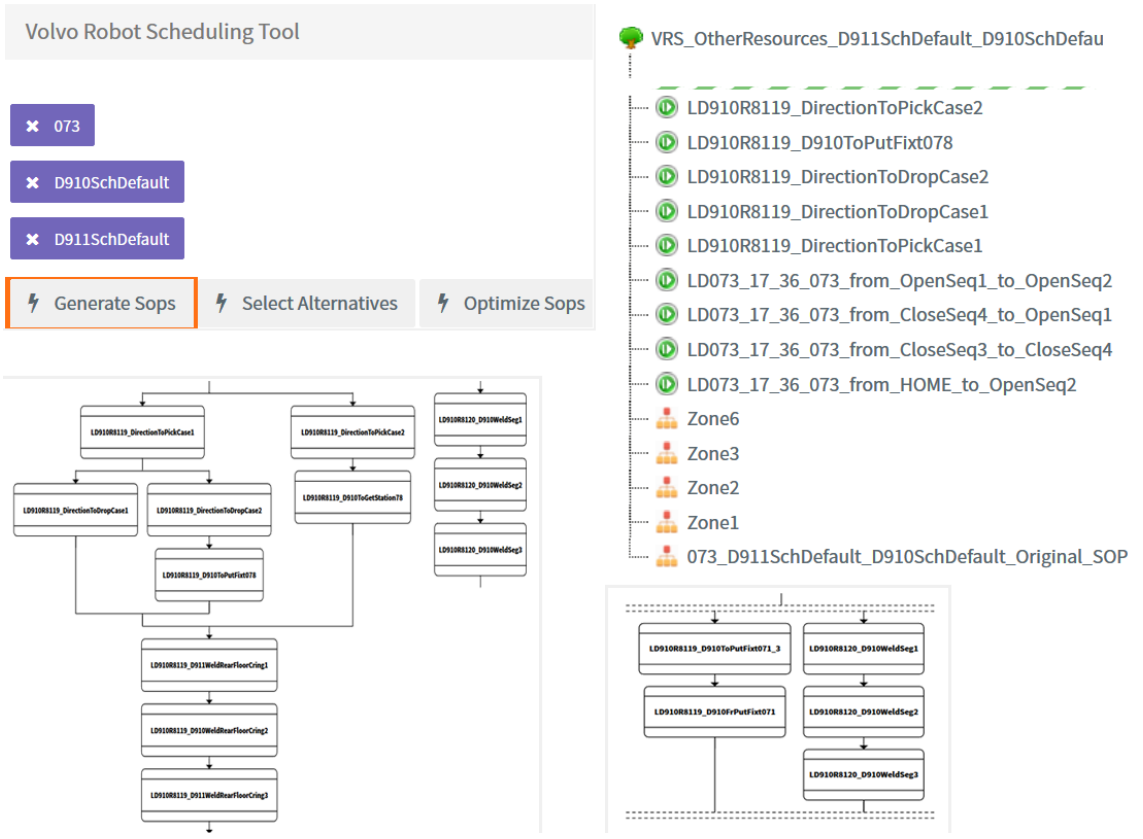

Figure 4.2: Generating SOPs for some resources. Top left are the selected schedules and resources that will be used for the generation, down low are the generated SOPs, and to the right the newly created hierarchy containing all the info.

4.3.1 Generating robot SOPs

Previously SP had been tested on robot cells which only contained robots, the robots having simple straight operation sequences. In our robot cell we encountered robots with alternative operation sequences. To handle this, it was necessary to change the program a bit, going into the alternatives and extracting their operations, adding them to the SOP. The zone handling also needed tweaking, to account for the alternatives.

To generate an SOP sequence from a robot program, the schedule of the program is needed. SP will read the schedule sequentially adding the operations to the SOP. However if an alternative is encountered, SP has to find the child operation sequences of the alternative and add them to the SOP recursively. When a zone is encountered in the schedule, it will either be allocated or released from a set of active zones. If an operation occurs within active zones, it will be added to a map of operations and zones, which is used to create zone SOPs. This is illustrated in Figure 4.3 and the pseudo code of Listing 4.3.

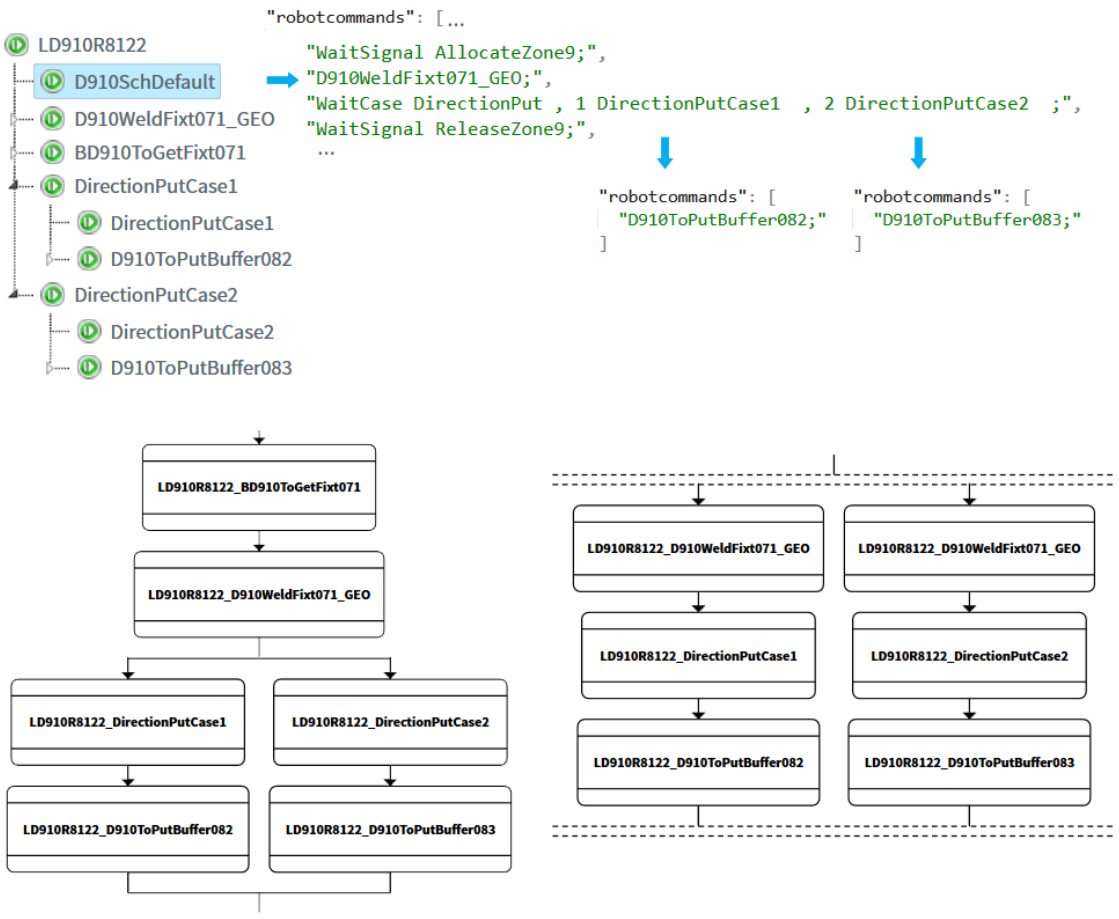

Figure 4.3: An illustration of how a part of the robot schedule D910SchDefault of robot 8122 is processed into SOPs, reading the schedule line by line and adding the operations to the SOPs.

Listing 4.3: Robot schedule SOP generation - pseudo code

```
function AddRobot(schedule)
    foreach (Command in schedule)

        if( Command == 'AllocateZone')
            Add the zone to a set of active zones

        else if( Command == 'ReleaseZone')
            Add mapped operations to the zone SOPs
            Remove zone from active Zones

        else if( Command == 'WaitCase')
            Get the operations of the Case
            Map the Case operation to active Zones
            if(Case contains new schedule)
                AddRobot(new schedule) // recursion
                Add new Sequences from Cases to main SOP
            Update the active zones and the zone map

        else
            Find an operation with the same name as the command
            Map operation to active zones
            Add operation to main SOP
```

We also corrected a problem where if a robot booked the same zone multiple times the zone specifications would become incorrect, putting all of the operations of one robot in the same sequence, when they should actually be separate from each other. A correct zone SOP is shown in Figure 4.4 and an incorrect SOP in Figure 4.5 for comparison.

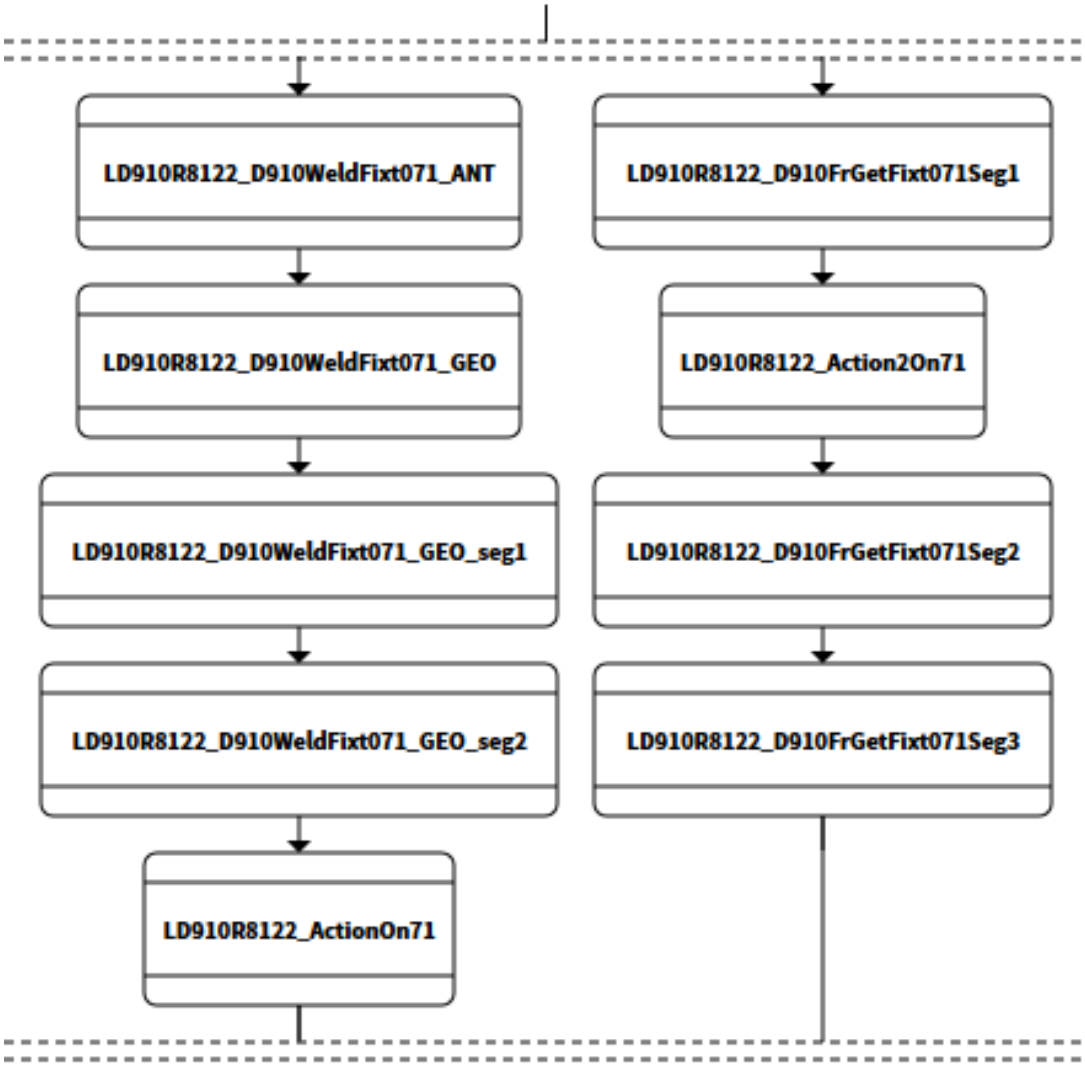

Figure 4.4: A zone that has been booked by the same robot at two separate times, adding those operations to different zone sequences, instead of the same one, so that if another robot wants to access this zone in between the current operation sequences, it can be granted access.

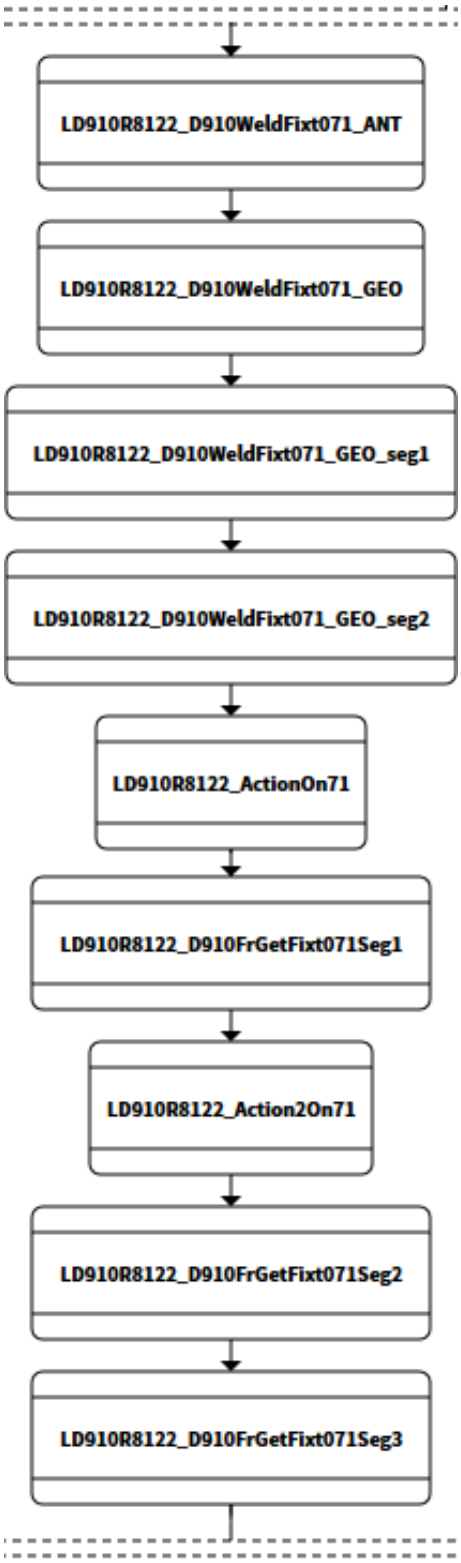

Figure 4.5: How SP incorrectly used to interpret a zone that had been booked by the same robot at two separate times, adding all operations to the same sequence.

4.3.2 Adding more resources to the model

The other moving resources of the cell - that are not robots - are typically controlled by the robot cell's PLC. This means that as in comparison to the robots there exists no easily understandable schedules or specifications for how they operate. In Process Simulate these devices are however modeled with the different poses i.e. positions they can take. From the poses in PS we can automatically generate all possible operations and signals between the poses.

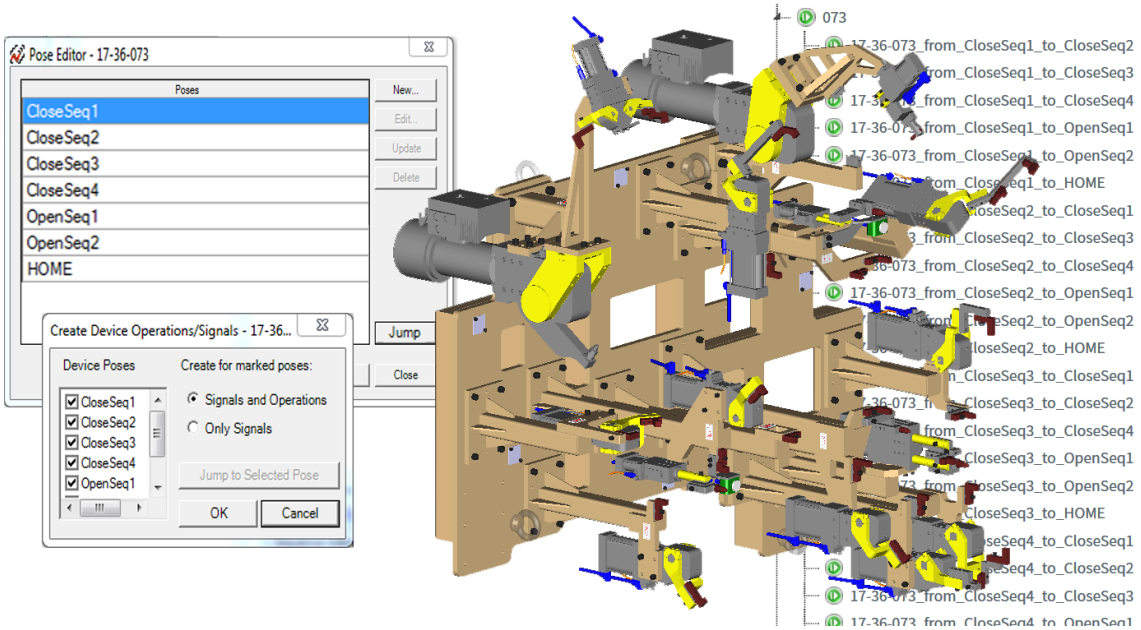

Figure 4.6: A turntable fixture in PS with poses. In the background some of the generated operations between poses, imported to SP.

These generated operations can be exported to Sequence Planner, but to know what to do with them and how a typical sequence looks for this type of device you need to know how it operates in the robot cell. The easiest way to understand how one of the resources operates is to get an understanding of how the cell as a whole works which is briefly explained in Section 1.3.

In TIA portal there is a section of the PLC program where SFCs (sequential flow charts) [27] for some of the resources, such as turntable fixtures, are stored. These do provide us with an idea of when the operations of the fixtures should be executed. In Figure 4.7 a fixture SFC is displayed.

Figure 4.7: An SFC for fixture 073 on turntable 072 in the cell from TIA portal.

After inspecting the SFC, PS model and robot programs, we can create a simplified specification of how the resource should work. For now we have only made a general model for one of the most common resource in the cell, namely the fixtures attached to the turn tables, see Listing 4.4 for the specification we created for this resource.

Each turntable has three fixtures, one for each car model. When the PLC has decided which model to produce, the turntables will interpose the correct fixtures towards the robots that moves and glues wheelhouses. The fixtures will secure the wheelhouse upon sensor, robot and PLC confirmation. Then the turntables will rotate to the middle of the cell, where welding operations can ensue after the slides which the turntables sits on are protruded to meet the middle fixture. There are several clamps to hold the parts together on different places during the welding process. Once done the slide will retract to allow the finished part to be extracted.

Listing 4.4: Turntable fixture sequences, simplified

```
Case 1
  1. Wait for the turntable to place fixture in load position
  2. Open all clamps and pins (OpenSeq2 in PS)
  3. Wait for robot 3 or 6 to place wheelhouse in fixture
  4. Set pins (CloseSeq1 in PS)
  5. Wait for robot 3 or 6 to request: close clamps
  6. Close clamps (CloseSeq2 in PS)
  7. Wait for robot 3 or 6 to clear the area
  8. Wait for fixture 071 to be loaded with the floor
  9. Wait for the turntable to place fixture in weld position

Case 2
  1. Wait for the slide of the turntable to move in
  2. Close additional clamps while robots are welding:
     (CloseSeq3, CloseSeq4, CloseSeq5(optional) in PS)

Case 3
  1. Release the part while welding (OpenSeq1, OpenSeq2 in PS)
  2. Wait for the slide to move out
```

We now know the order of the poses, and can define this order in SP for this type of resource. Using the list of poses, we can find the right operations between the poses and create an SOP sequence out of them according to the pseudo code in Listing 4.5. Since the cases of the SFC should ideally be executed after each other, we assume this order in the SOP generation. For an example of an SOP created in this way, see Figure 4.8.

Listing 4.5: Pseudo code for adding a fixture to the SOP model

```
function AddOtherResource(resource)
    if(resource =='TurntableFixture')
        currentPose = HOME
        poses = List(OpenSeq2,CloseSeq1,.... )
    for(nextPose in poses)
        find operation with startPose= currentPose & endPose= nextPose
        update currentPose with nextPose, and add operation to a list

    Create an SOP sequence out of the operation list
```

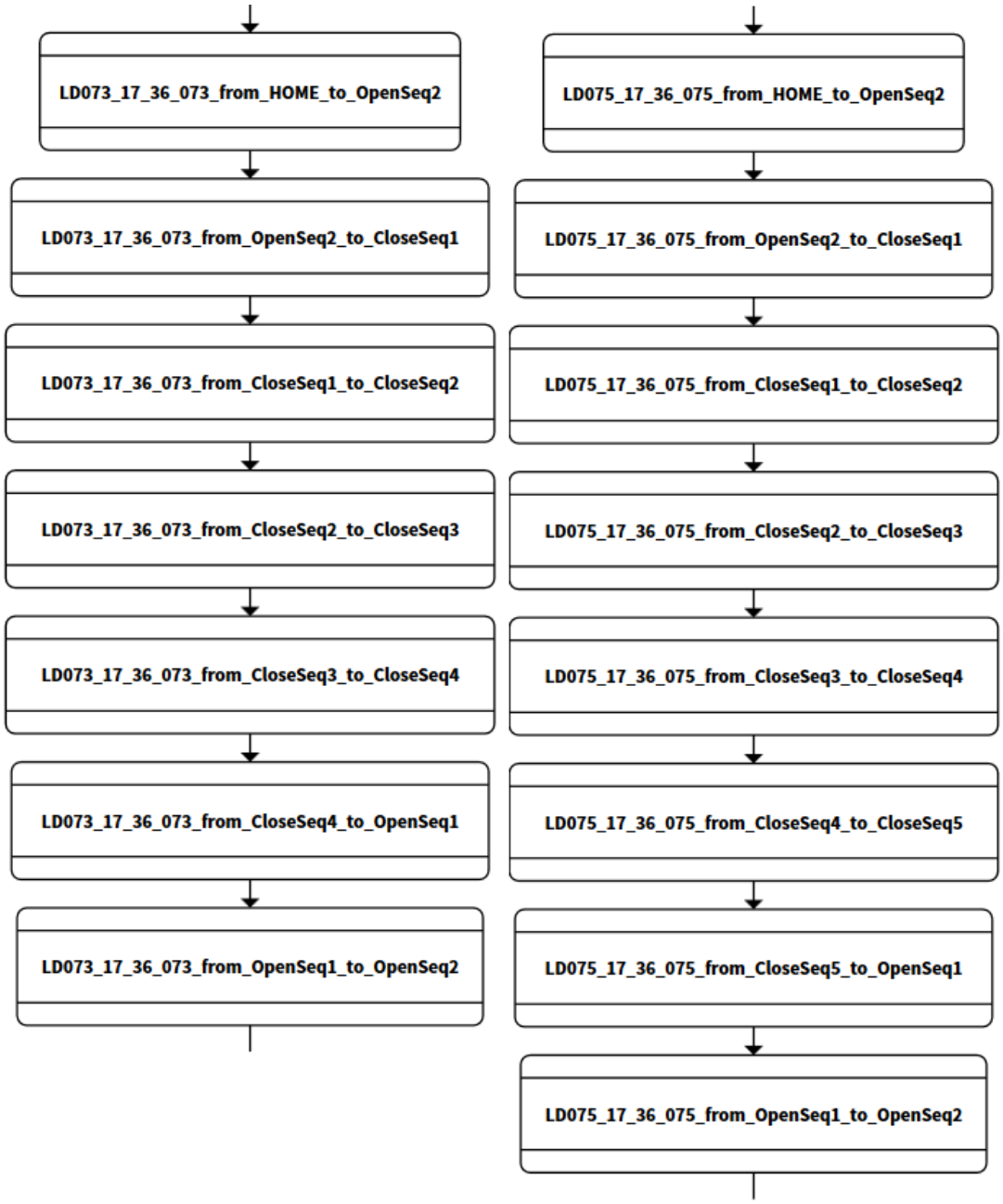

Figure 4.8: The generated SOP for the turntable fixtures 073 and 075.

This way of adding resources could easily be extended to handle more resources.

However, even though the resources have been added to the model, the logical behaviour between resources specified by the PLC is not incorporated.

4.4 Creating a new SOP model for the PLC logic

When a good understanding of the PLC and robot cell has been acquired it should be modeled in SP. The most important information sought is the fixed order of operations between resources.

For instance, before the turntables are allowed to move to the weld position, the middle fixture has to be loaded, and the robot loading that fixture has to be out of that area. And before any weld operations can commence, the right fixtures should be loaded and in position.

All these operation dependencies should be manually modeled in a separate SOP which is used to create constraints for the optimization.

We did not model the entire PLC and robot cell, but for future reference we provide some brief notes on how to gain some of the required information to this end in Appendix A.2.

We did however enable the creation of a PLC SOP, where the user can manually model the dependencies between resources.

An example of how this SOP can be used is shown in Figure 4.9 below. Where the dependency that robot 1 (8119) has to have placed the car floor in the middle fixture (071) and vacated the area before robot 2 (8120) can start welding is modeled.

The impact of this model on the optimization is shown in the next section.

For now this SOP only provides information for the constraints of the CP optimizer. These operation dependencies could also be added to the supervisor in the future.

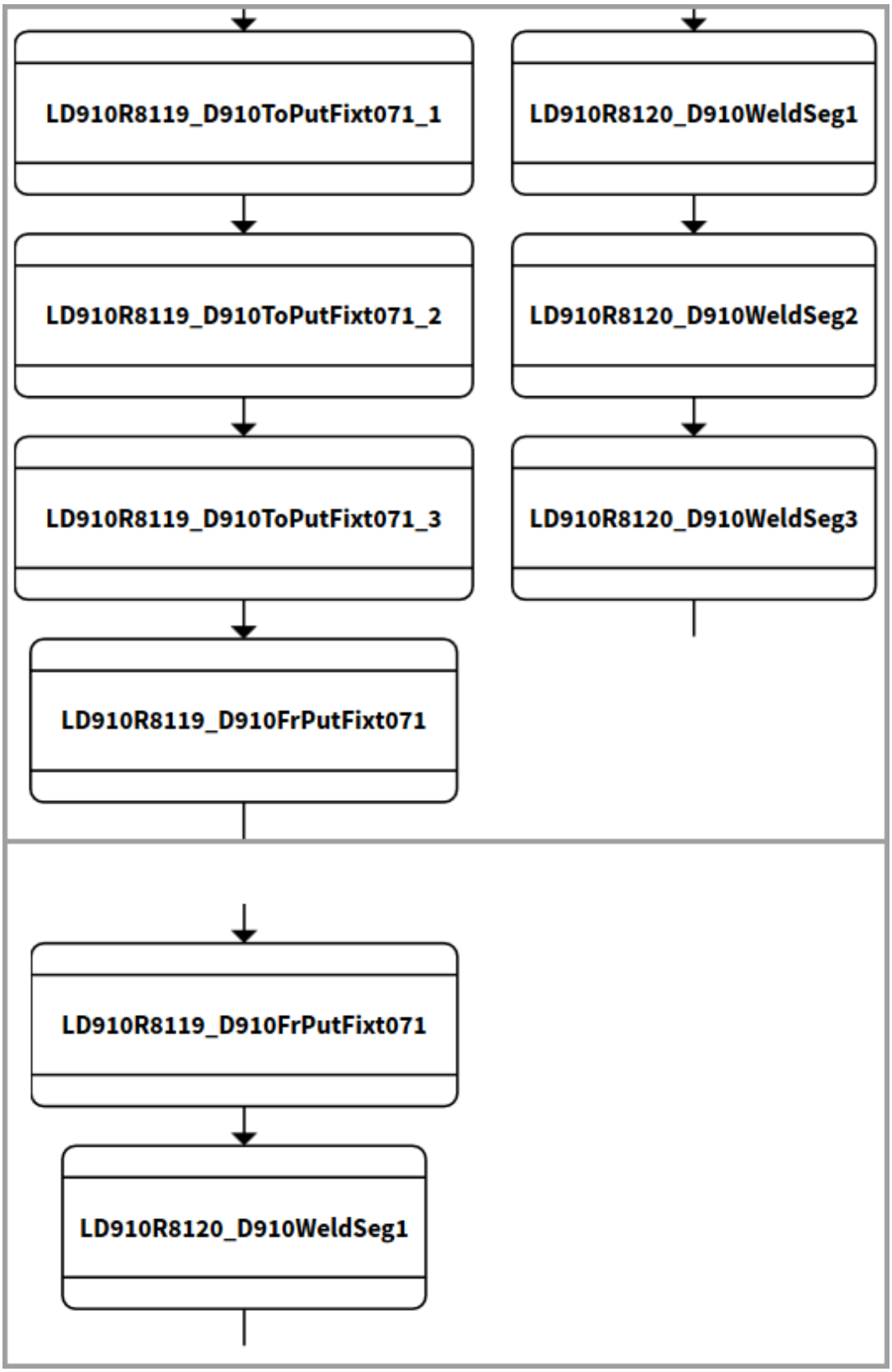

Figure 4.9: In this figure, part of the R8119 sequence is displayed together with R8120. On the lower part of the figure, an SOP has been created to model the PLC dependencies between the operations. So that R8119 has returned from fixture 071 before R8120 can start welding.

4.5 Synthesizing and optimizing

When the model is complete, it can be synthesized and optimized in the same step, as in Listing 4.6.

Listing 4.6: Synthesizing and optimizing, procedure.

```
Get all the SOPs and operations from the created SP model.
Sort out the operation Sequence SOP, the PLC SOP and the Zone SOPs.

Use the Sequence and Zone SOPs for synthesis.

From all of the SOPs and with the selected cases,
filter out the possible operation sequences
and create constraints for the optimization.
```

4.5.1 Synthesizing a supervisor

The synthesis is performed by Supremica, which will look at the SOP operation sequences and form guards between the operations, given their order in the SOP and the zone specification SOPs, yielding results as in Figure 3.1.

The model has to be preprocessed before synthesis, adding information about transitions between operations, and information about zones. Here we had to modify the code to handle alternative branches, enabling several operations transitioning into the same operation. See Figure 4.10 and Listing 4.7 for the transition conditions.

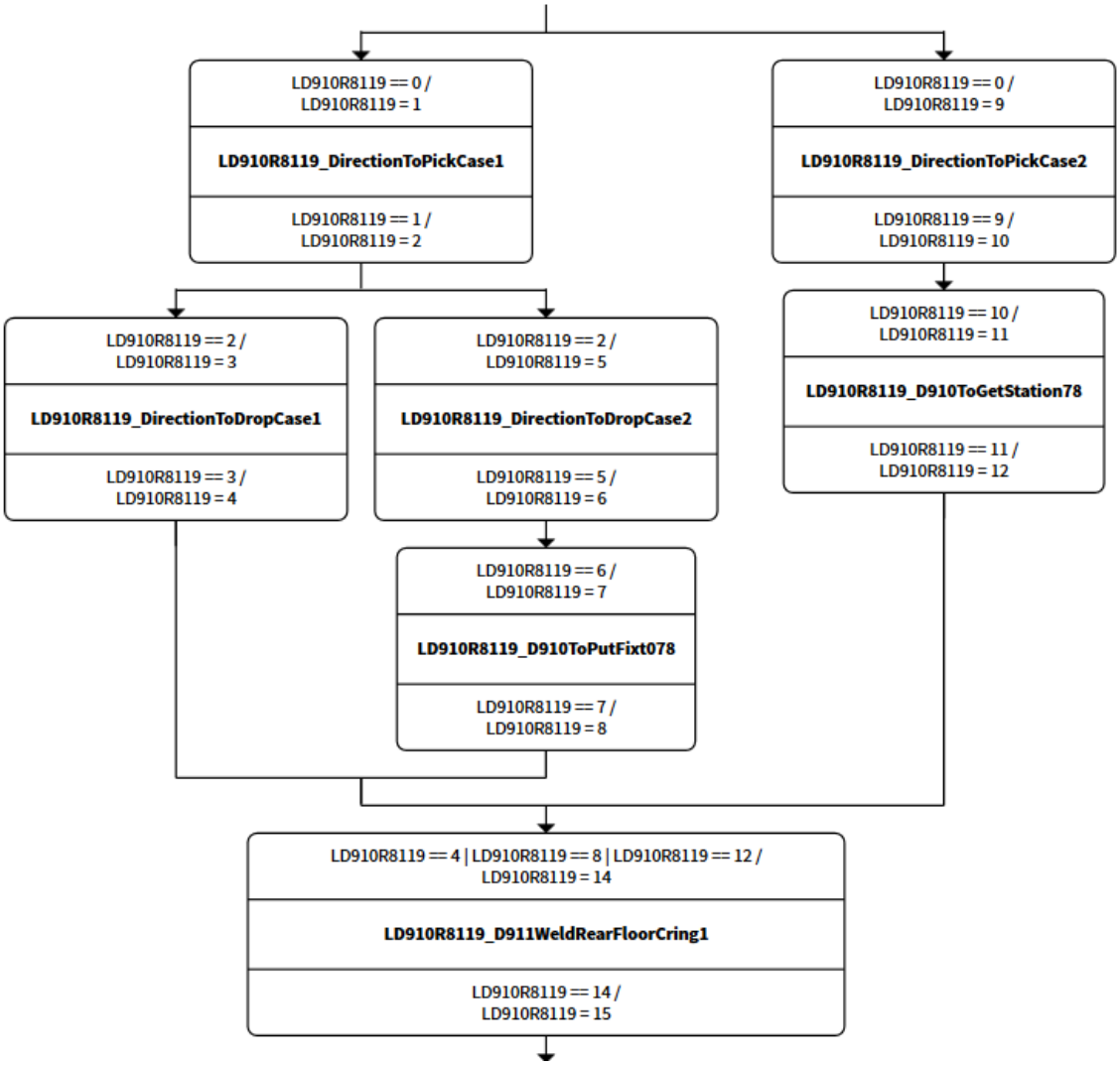

Figure 4.10: An SOP for robot 19 with synthesized guards from the sequence of operations.

Listing 4.7: For the SOP in Figure 4.10 a transition condition will be added to operation D911WeldRearFloorCring1 , where one of the operation's prequels has to be finished before it can start

```
atStart:    LD910R8119_DirectionToDropCase1_done OR
            LD910R8119_D910ToPutFixt078_done     OR
            LD910R8119_D910ToGetStation78_done

atExecute:  LD910R8119_D911WeldRearFloorCring1

atComplete: LD910R8119_D911WeldRearFloorCring1_done
```

When the preprocessing is done the model will be sent to Supremica, where a non blocking, controllable and maximally permissive supervisor is created, from which the synthesized guards are extracted and added to the operations.

4.5.2 Optimizing the sequences

Since the CP problem is currently not setup for alternative sequences, it is necessary for the user to select which cases should be active during the optimization. This will make sure that only the chosen operation sequences will be passed to the solver. See Figure 4.11 for an illustration, where the selected cases will be sent back to the scheduling tool, which will filter out the correct sequences from the SOPs.

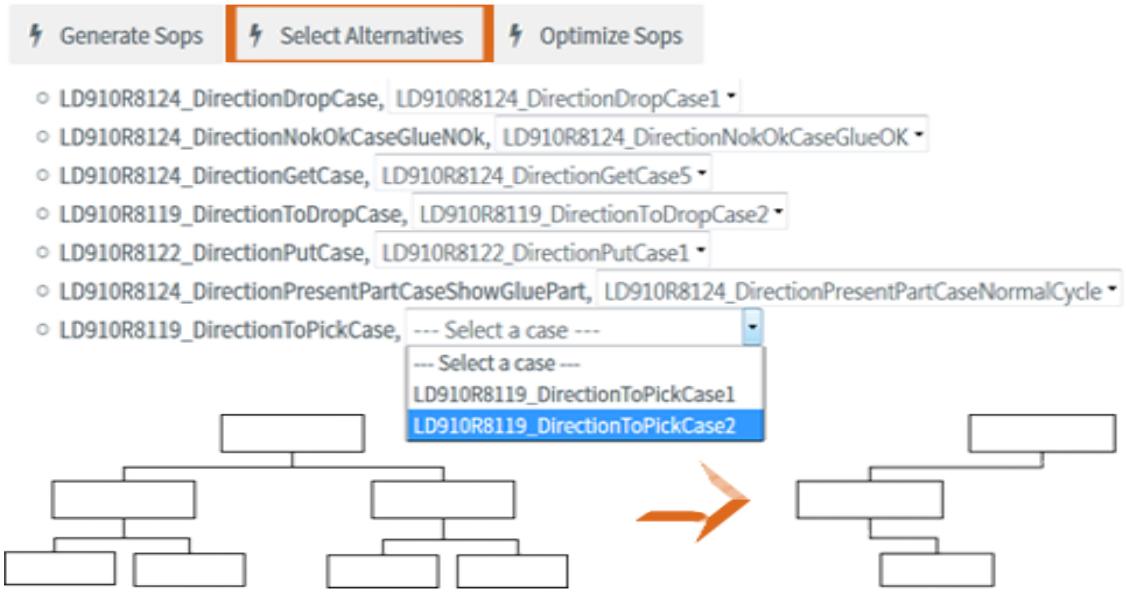

Figure 4.11: Selecting alternative branches, to create straight operation sequences for the optimization.

Before the optimization can commence, the model has to be formulated as a mathematical problem which the constraint programming algorithm can solve. The selected SOP sequences and PLC SOP sequences give CP constraints on the order in which the operations should execute, and the zones provide constraints on which operations that cannot execute at the same time. This information, in conjunction with the execution times of the operations is sent to the CP solver, which will solve the problem as in Section 3.1.4.

With the PLC SOP from the previous Section 4.4 in Figure 4.9, constraints are added to the optimized SOP, preventing robot 2 (8120) from starting before robot 1 (8119) is out of fixture 071 as shown in Figure 4.12.

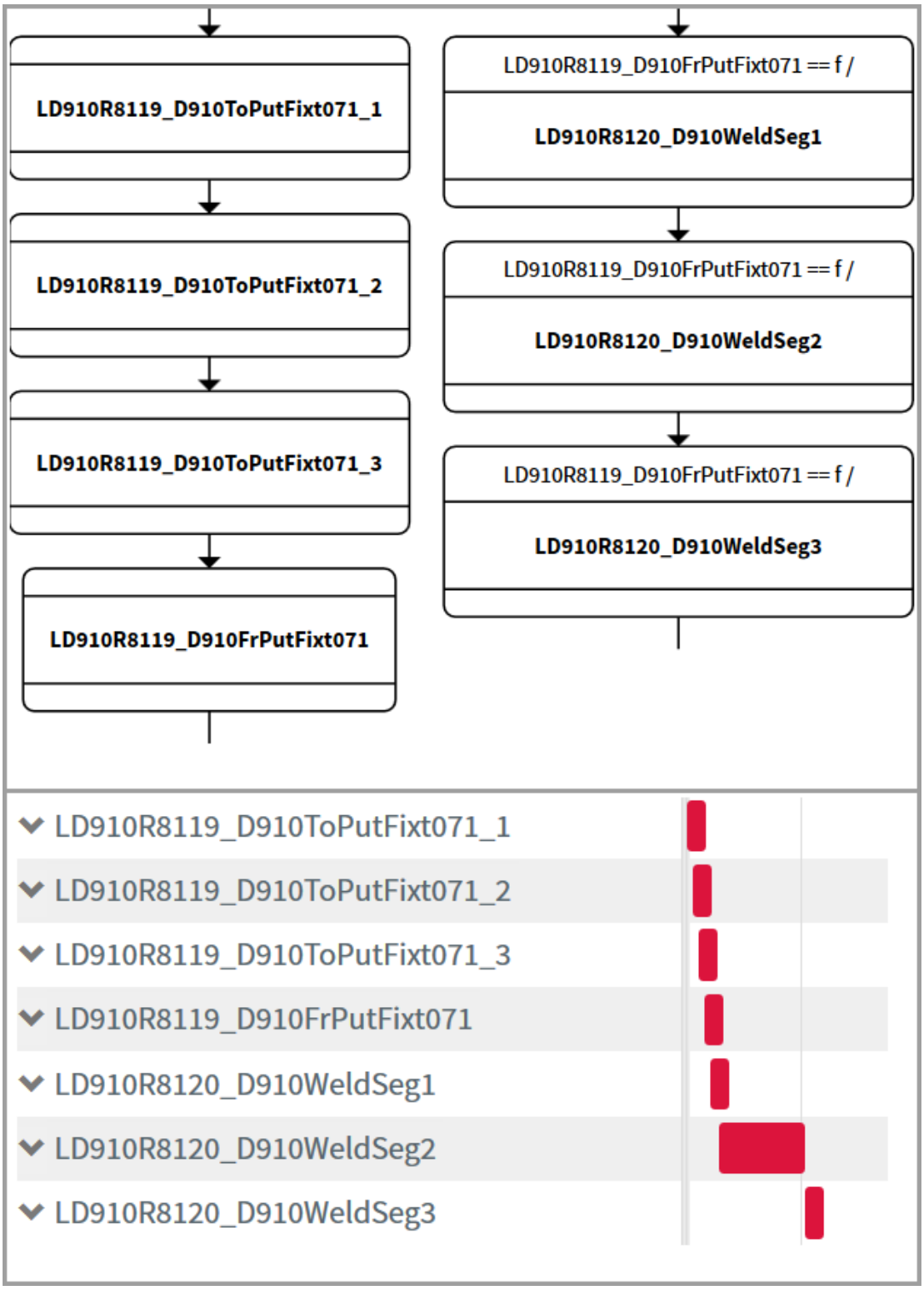

Figure 4.12: Optimizing the SOPs in Figure 4.9, creating a new SOP with added constraints on the execution order, giving the sequence shown in the gantt chart.

4.6 Exporting to Process Simulate

Once optimization is complete, the user can select which of the SOP solutions they want to export back to process Simulate for virtual verification and testing as before. The SOP will be translated back into PS operations connected with transitions, mirroring the optimized sequence such as that in Figure 4.12. Since we could not get the PS model running properly, due to missing signal mapping, we have not verified any sequences there.

5
Results

The Volvo robot scheduling tool in Sequence Planner can now import both robot programs with alternative branches as well as other resources than robots through the plugin described in Section 4.1. It can automatically model all robots and some of the other resources using SOPs. After automatic modeling, the user can manually change or add to the model, incorporating more of the robot cell's and PLC's behaviour. Once the modeling is complete, synthesis and optimization can ensue. The synthesis now also works for alternative sequences and will make sure that the system is minimally restrictive, controllable and non-blocking, however the PLC SOP model is not yet incorporated with the synthesis. For the optimization, which currently cannot handle alternatives, the user has to select which alternatives should be active during this step. After constraint programming the solution sequences can be exported back to PS for visual verification and testing as before.

5.1 Discrepancies

5.1.1 BDD verification

The BDD verifier can check which operation transition conditions that can and cannot be fulfilled at the same time. Each operation has three transition conditions, one which should be fulfilled before it starts i.e. a guard, one when it is executing the actual operation and one saying that the operation is done, the completion condition. The problem is that now that we have added alternatives, a transition guard may consist out of several operation ending conditions as a long text string. The BDD verifier does not know how to interpret this. The solution would be to split the string into its separate parts and run the BDD verifier for each of them. This is not a significant problem, since it is possible to manually check each completion condition but not all of them at the same time. In Figure 4.10 the operation D911WeldRearFloorCring1, where the guard consists of three conditions would give this error.

5.1.2 Synthesis

The synthesis function seems to have a problem with operation names starting with numbers and names containing the symbol dash -. We are not entirely sure where this problem originates but it should be looked into. For now we simply modify the operation names to comply with the synthesizer.

6
Conclusion and Discussion

Overall it was a nice project, we learned a lot and managed to pave the way for a brighter, more autonomous future. Below is an evaluation of the current work and suggestions of improvements, as well as an analysis of how this work can impact the environment in a positive way.

6.1 Project evaluation and future work

6.1.1 Sequence Planner

Sequence planner is a work in progress, developed by researchers and students, as such there is not always documentation at hand. We have tried to explain implementations here and in the actual code to make it a bit easier to follow in the future and we would urge everyone working with SP to do the same.

6.1.1.1 Optimization

Currently it is possible to optimize each alternative sequence at a time; this works well, but it would be nice if it was possible to optimize all alternatives together, saving time and making it easier to compare the results. To do this, the problem has to be formulated in a way so that the CP solver handle the alternative branches correctly.

The SOPs with alternatives and added conditions would have to be exported to PS. But upon trying to export SOPs with alternatives, we concluded that there seems to be a lack of support for adding alternatives with conditions using the PS application programming interface (API). It was only possible to add parallel sequences as a standard. This would have to be worked out before such an optimization approach is considered.

6.1.2 Adding resources to the model and modeling the PLC

It was quite difficult to get a good grasp of how the PLC program worked, since there were no good specifications or explanations, as well as a lack of standard naming convention between the PLC and robot programs.

The PS model was also lacking functionality, since the resources did not have connected signals. The operations in PS did not have a defined execution time either, which is necessary for a proper sequence optimization. This can however be gathered from running the PS model, if the signals are connected.

To make it easier in the future, there has to be more cooperation between the teams working on the PLC, robots and modeling. The PS model should be workable without knowing how the entire PLC code is functioning. It would be beneficial if there was a simple specification for how the whole cell operates as well as the individual resources.

As for SP, it would be nice if the user could define new general resources through the GUI, not having to go into the program code and hard coding them.

6.1.3 Combining this and similar work

We mentioned earlier that this thesis was part of a group of closely related projects whitin VirtCom. Due to compatibility issues and limited time the projects could not merge successfully.

The project about virtual commissioning resulted in suggestions on how Volvo should improve their methodology when performing virtual commissioning (to ensure that the virtual models are more user friendly) and a program for automatically mapping PLC signals to the PS model. This information should be used by Volvo in their coming VC projects and would enable them to create proper models, for us to use in the sequence optimization. For now, the mapping program can be used when trying to model resources and the PLC in SP.

The project concerning automatically creating zones between robots in PS by generating volumes where the robots could collide, was destined to provide us with updated robot schedules, containing their created zones. However upon zone creation, they ended up adding that information to off-Line Programming (OLP) commands in PS, instead of to the robot schedules which we use in our program. To use the information, their program would have to write information to the robot schedules, or SP would have to be extensively modified to interpret OLP commands.

6.2 Sustainability

By introducing tools such as Sequence Planner, it will become easier to find time optimal and deadlock free sequences for the engineers, allowing them to direct their focus elsewhere.

Optimal sequencing can reduce, downtime and waiting times in the robot cells, making it less likely for the robots to cool down while waiting for other processes. A cold robot may lack in precision and draw more energy, due to the robot's material temperature distortion [28] and higher mechanical friction [29]. Therefore an additional warm up time may be required where the robot cannot perform any meaningful work, which consumes more energy.

The production rate also increases with good sequencing, cutting down on time the factory needs to be up and running. Another advantage is that this approach could enable the factories to reduce the amount of concurrently working operators, provided that fewer errors occur.

Bibliography

[1] E. Ohlson and C. Thorstensson, "Development, implementation and testing of sequence planner, a concept for modeling of automation systems," Master's thesis, Chalmers University of Technology, 2009.

[2] A. Bockmayr and J. N. Hooker, "Constraint programming." Carnegie Mellon University, May 2003.

[3] P. J. G. Ramadge and W. M. Wonham, "The control of discrete event systems," *Proceedings of the IEEE*, vol. 77, pp. 81–98, Jan 1989.

[4] A. Pfeiffer, B. Kádár, and L. Monostori, "Evaluating and improving production control systems by using emulation," 2014.

[5] M. Dahl, K. Bengtsson, P. Bergagård, M. Fabian, and P. Falkman, "Integrated virtual preparation and commissioning: supporting formal methods during automation systems development," in *IFAC-PapersOnLine*, pp. 1939–1944, 2016.

[6] S. Makris, G. Michalos, and G. Chryssolouris, "Virtual commissioning of an assembly cell with cooperating robots," *Advances in Decision Sciences*, vol. 2012, pp. 1–11, August 2012.

[7] B. Lennartsson, M. Fabian, and K. Åkesson, "Introduction to discrete event systems," 2009.

[8] K. Åkesson, M. Fabian, H. Flordal, and R. Malik, "Supremica - an integrated environment for verification, synthesis and simulation of discrete event systems," *Proceeding os the 8th Workshop on Discrete Event Systems (WODES'06), Ann Arbor, MI, USA*, vol. s. 384-385, 2006.

[9] M. Dahl, K. Bengtsson, P. Bergagård, M. Fabian, and P. Falkman, "Sequence planner: Supporting integrated virtual preparation and commissioning," in *The 20th World Congress of the International Federation of Automatic Control, 9-14 July 2017*, 2017.

[10] S. Riazi, K. Bengtsson, R. Bischoff, A. Aurnhammer, O. Wigström, and B. Lennartson, "Energy and peak-power optimization of existing time-optimal robot trajectories," in *2016 IEEE International Conference on Automation Science and Engineering (CASE)*, pp. 321–327, Aug 2016.

[11] A. Theorin, K. Bengtsson, J. Provost, M. Lieder, C. Johnsson, T. Lundholm, and B. Lennartson, "An event-driven manufacturing information system architecture for industry 4.0," *International Journal of Production Research*, pp. 1–15, 2017.

[12] N. Dragoni, S. Giallorenzo, A. Lluch-Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina, "Microservices: yesterday, today, and tomorrow," *CoRR*, vol. abs/1606.04036, 2016.

[13] "Sequence planner." Available from: `https://github.com/kristoferB/SP`, 2017-02-06.

[14] B. Lennartson, K. Bengtsson, C. Yuan, K. Andersson, M. Fabian, P. Falkman, and K. Akesson, "Sequence planning for integrated product, process and automation design," *IEEE Transactions on Automation Science and Engineering*, vol. 7, pp. 791–802, Oct 2010.

[15] M. Skoldstam, K. Akesson, and M. Fabian, "Modeling of discrete event systems using finite automata with variables," in *2007 46th IEEE Conference on Decision and Control*, pp. 3387–3392, Dec 2007.

[16] K. Bengtsson, P. Bergagård, and C. Thorstensson, "Sequence planning using multiple and coordinated sequences of operations," 2012.

[17] M. Sköldstam, K. Åkesson, and M. Fabian, "Supervisory control applied to automata extended with variables - revised," tech. rep., 2008. 14.

[18] S. B. Akers, "Binary decision diagrams," *IEEE Transactions on Computers*, vol. C-27, pp. 509–516, June 1978.

[19] N. Sundström, O. Wigström, P. Falkman, and B. Lennartson, "Optimization of operation sequences using constraint programming," in *IFAC Proceedings Volumes. 14th IFAC Symposium on Information Control Problems in Manufacturing, INCOM'12, Bucharest, 23-25 May 2012*, pp. 1580–1585, 2012.

[20] J. Zhou, "A constraint program for solving the job-shop problem," in *In Second International Conference on Principles and Practice of Constraint Programming (CP'96*, p. 150, Springer Verlag, 1996.

[21] OscaR Team, "OscaR: Scala in OR," 2012. Available from: `https://bitbucket.org/oscarlib/oscar`.

[22] "Process simulate, manufacturing process verification in powerful 3d environment." Available from: `https://w5.siemens.com/italy/web/AD/MECSPE/Documents/Siemens-PLM-Tecnomatix-Process-Simulate-fs-X7-2.pdf`, 2011.

[23] A. Katzenbach, S. Handschuh, and S. Vettermann, "Jt format (iso 14306) and ap 242 (iso 10303): The step to the next generation collaborative product creation," pp. 41–52, Springer Berlin Heidelberg, 2013.

[24] "Process Simulate on eMS Intermediate Robotics (CEE)." Publication Number: MT42215-S-121, MT42215-version 12.1, September 2015. Student Guide.

[25] N. Sundström, "Automatic generation of operations for flexa production system," 2010.

[26] "Totally Integrated Automation Portal: One integrated engineering framework for all automation tasks." `https://www.automation.siemens.com/salesmaterial-as/brochure/en/brochure_tia_portal_en.pdf`, 2013. Accessed: 2017-08-24.

[27] "Open international standard iec 61131-3, languages for programmable logic controllers," 2013.

[28] P. Poonyapak, *Improving Robot Efficiency to Reduce Energy Consumption.* ICWES14, 2008.

[29] S. Riazi, K. Bengtsson, O. Wigström, and B. Lennartson, "Energy and peak-power optimization of existing time-optimal robot trajectories," in *Proc. 12th IEEE Conference on Automation Science and Engineering (CASE 2016), Fort Worth, Texas, August*, pp. 321–327, 2016.

A

Appendix 1

A.1 Tips and tricks

A.1.1 Setting up Sequence Planner at Volvo

When working with the software Sequence Planner it is necessary to have internet access to the Git hub repository during setup. Therefore if working at a company such as Volvo Cars where internet access is restricted, it may be (as in our case) necessary to create a virtual machine (VM) on which to run all Sequence Planner related software.

We used Virtual box to install a Linux distro on which to run SP, active mq (for the communication between SP and PS) and IntelliJ.

To connect the VM to internet at home and to allow communication between the VM and the host we configured the network connections as in Figure A.1. When importing a new VM it was also important to reinitialize the MAC addresses for everything to work properly.

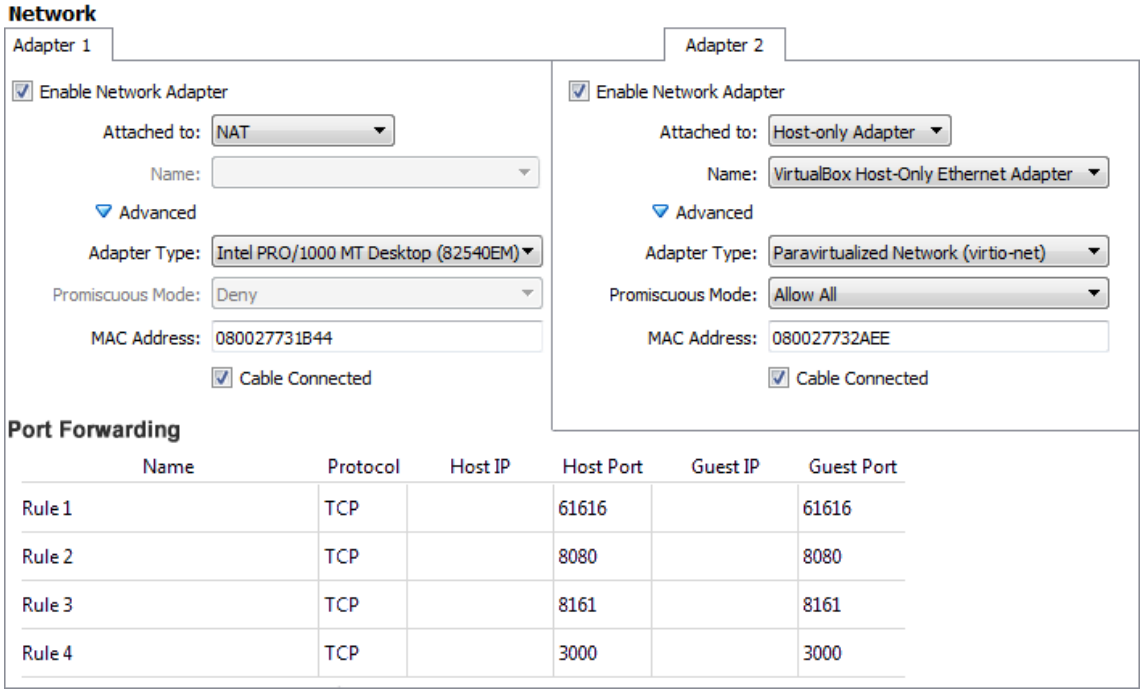

Figure A.1: How our network and port forwarding between the virtual machine and the host machine was setup for SP and active mq.

The distro we selected was LUbuntu standing for light Ubuntu, it is definitely not the lightest Linux distro out there, but reletivly simple to install. Performance wise it would be better to opt for something like Arch Linux.

IntelliJ is one of the most reputable Scala IDEs on the market which is why we chose that as our programming tool. Through IntelliJ it is possible to first install a Scala add on and then download the whole Sequence Planner project directly and selecting a desired branch. It also enables the user to use the display type option to show types of different variables. Although IntelliJ has a lot of good coding features it also requires allot of your hardware.

Once Sequence Planner is downloaded it is necessary to install some additional tools following the instructions in the SP main folder and the SP/gui/web folder, for the old version of SP.

To allow communication with Process simulate and the plugin, active-mq is required and we installed it on the VM to run at startup.

A.1.2 Debuging the PS plugin

The plugin is written in *C#* and so we used Visual Studio to modify it. In Visual Studio it is possible to debug the program by attaching it to the PS process which is called Tune, for some reason.

A.2 Modeling the PLC and operation relations

We have not modeled the PLC in SP although some efforts in this direction were made, which is why this chapter is provided as guidelines for eventual future attempts, since it is a crucial step towards optimizing the whole robot cell.

Good knowledge of how both the PLC program and robotprograms are working is required to replicate the behaviour in an SP model. To this end it is also desirable to have specifications for each resource, a signal mapping between PS and the PLC and a common naming convention.

Specifications for the robot cell were not available, so an understanding of how it worked had to be inferred from the robot programs, PLC code and PS model.

In TIA portal the PLC programs can be examined. Each robot has a HMI (Human machine Interface) as in Figure A.2, where some of the signals connected to the robot are listed. From there the signals origin can be backtracked, and matching signals can be found in PS or the robot programs.

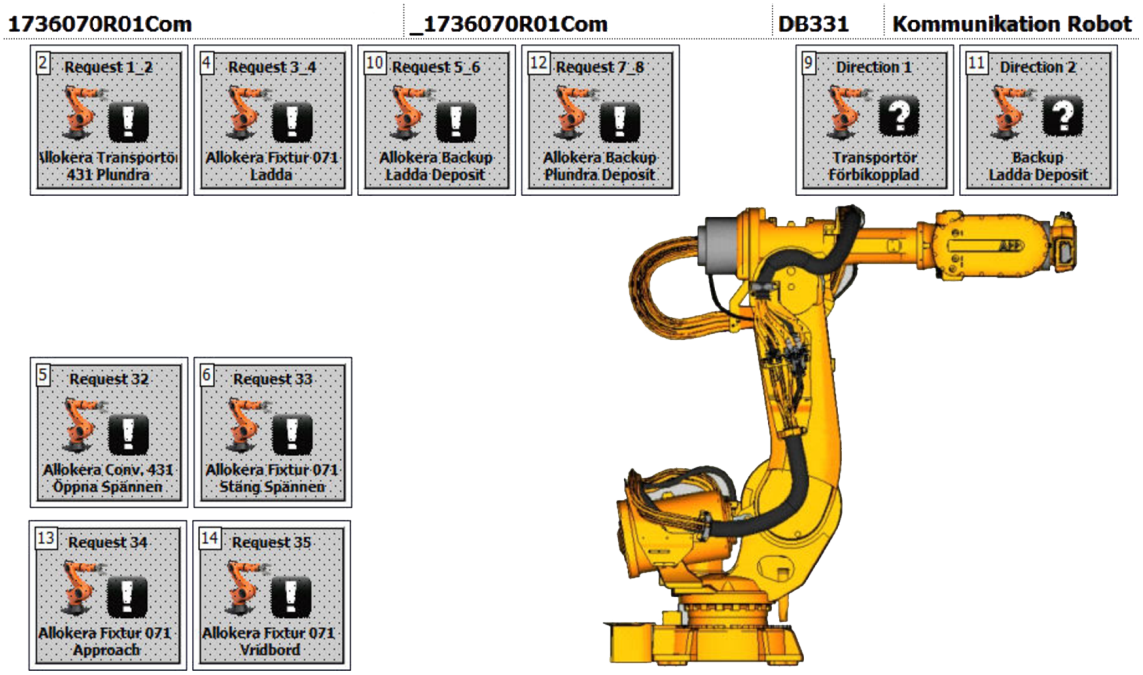

Figure A.2: An HMI in TIA portal for the robot communication, showcasing some of the signals which can also be found in the robot program.

In the robot programs there is a section which maps PLC signals to variable names, see listing A.1, used in the robot schedules. Each robot has its own local signals in the PLC, meaning that several robots can have the same variable names and PLC numbers associated to them in their respective PLC signal map. However there is a special section where the common zones are stored in the PLC and they have fixed global PLC signal numbering,

Putting all of the information together from robot programs, PLC -logic blocks, -ladder rungs, -SFCs, -HMIs and the PS model it should be possible to model the relations between different operations and resources.

Listing A.1: Robot schedule LD910R8119 PLC signal map definitions, sample -prettified

```
   AllocateStation1      [1, Allocate Conveyor 17-36-431]
   ReleaseStation1       [2, Release Conveyor 17-36-431]

   AllocateZone1         [21, Allocate Zone1 Robot8120]
   ReleaseZone1          [22, Release Zone1 Robot8120]

   ActionOnConv          [32, Allocate Conveyor 17-36-060, open clamp]
   ActionOn071           [33, Allocate 17-36-071 and close clamps]

   WorkIsDone            [31, Work is done]

   DirectionToDrop       [2, Drop on fixture 071 or St78 buffer]
   DirectionToPick       [1, Pick from Conveyor or St78]
   ...
```

A.3 Robot cell

The robot cell's naming of components from PS is presented in Figure A.3 bellow. As you can see this picture is a bit more detailed than before, to lift out the different components. For the robots we have include both the PLC naming counterclockwise from robot 1-7, and the PS naming 81(19-25), note that the numbering on robot 4 and 5 are flipped in PS.

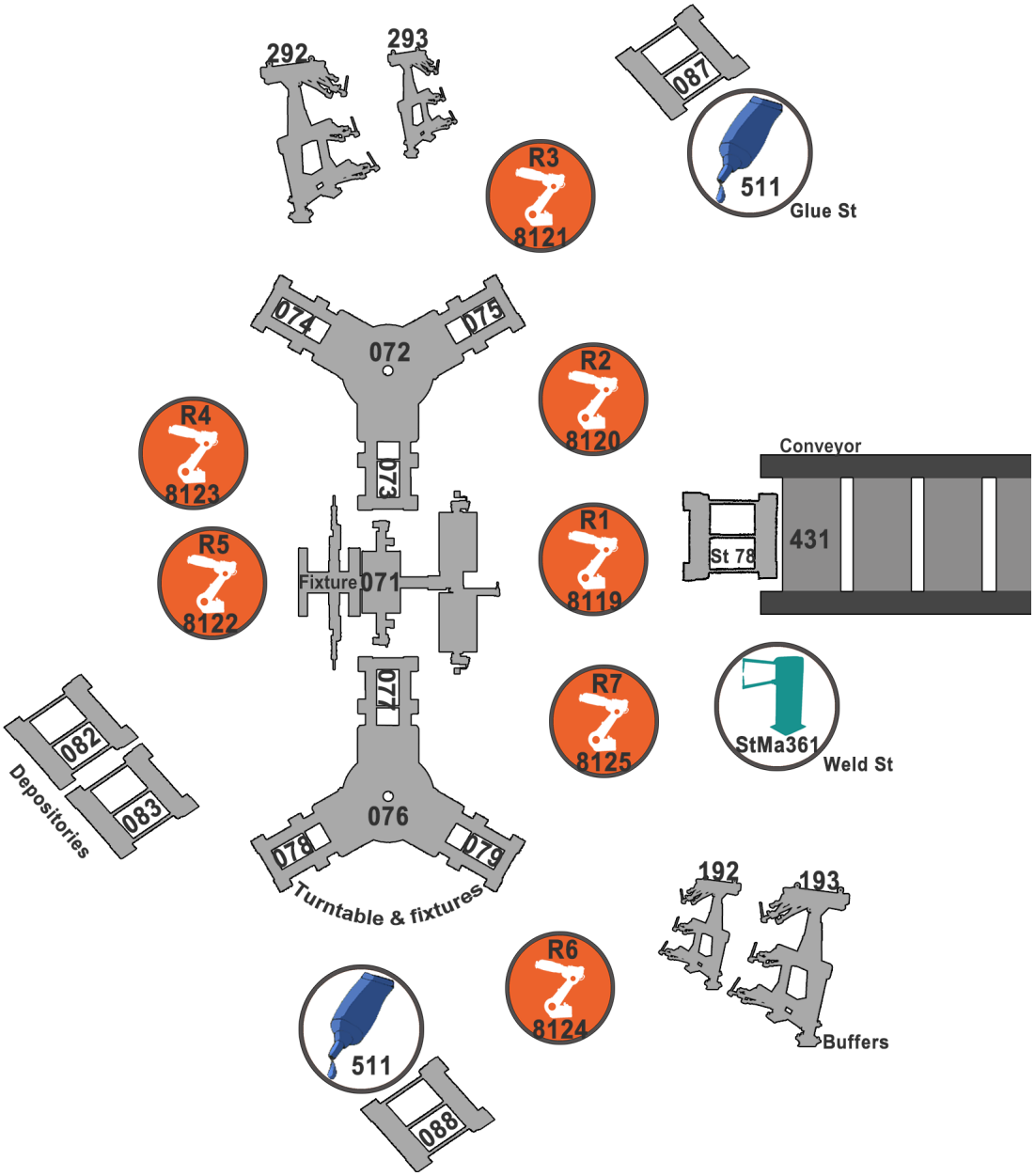

Figure A.3: A more detailed overview of the robot cell, with component names included.