

**MASTER**

**Interoperability between LSAT and CIF**

Chakraborty, Saikat

*Award date:*  
2021

[Link to publication](#)

**Disclaimer**

This document contains a student thesis (bachelor's or master's), as authored by a student at Eindhoven University of Technology. Student theses are made available in the TU/e repository upon obtaining the required degree. The grade received is not published on the document as presented in the repository. The required complexity or quality of research of student theses may vary by program, and the required minimum study period may vary in duration.

**General rights**

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain

# ASML

DEPARTMENT OF MECHANICAL ENGINEERING  
CONTROL SYSTEMS TECHNOLOGY GROUP  
MASTER: MANUFACTURING SYSTEMS ENGINEERING

---

## **Interoperability between LSAT and CIF**

---

**Author:**

Saikat Chakraborty [1413961]

**Thesis Supervisor:**

Dr. Ir. M.A. Reniers

**Supervisor:**

Ir. S. B. Thuijsman

**Report ID:**

CST2021.064

**Eindhoven, September 28, 2021**

---

## Abstract

With the rapid growth of complex manufacturing systems, the importance of understanding, analysis, and simplification of these systems from an engineering perspective has also risen. One approach that is widely accepted to be highly effective in this regard is Model Based Engineering (MBE). It is an approach that uses models as an integral part of the technical baseline that includes the requirements, analysis, design, implementation, and verification of a capability, system, and/or product. Generally, a model is defined as a simplified conceptual representation of a given system that can be used to improve ease of understanding, simulate combination(s) of various possible events, and predict scenarios likely to occur.

Various tools and languages have been developed for MBE. E.g., Supremica, Compositional Interchange Format (CIF) and Logistics Specification and Analysis Tool (LSAT) are some of the popular ones available. The contents of this research will be focused on CIF and LSAT only.

LSAT is an Eclipse project developed and maintained jointly by ASML, ESI (TNO), and Eindhoven University of Technology, which enables users to design flexible manufacturing systems while complying to the philosophy of other MBE design tools. The functionality of the toolkit includes, but is not limited to, the specification of a system and the relevant product flow within it, analysis of the resources being used in the associated processes and optimization of the resource usage based on the results of the analysis. CIF, on the other hand, is a more general tool used in the MBE domain. Developed and maintained by Eindhoven University of Technology, it is an automata-based modeling language wherein an automaton is used to describe a discrete-event, timed or hybrid system. An automaton itself, in its most elementary form, consists of possible states the system may achieve and events to transition to and from various states. Additionally and more importantly, CIF enables the synthesis of (supervisory) controllers which govern a given system adhering to certain requirements as defined by the user.

The primary objective of this research is to describe how using an automata representation of a system specified in LSAT and by using specific supervisory controller synthesis techniques as described in this research, controllers can be synthesised depending upon the required degree of control granularity specified by the user.

Besides that, the second aspect of this research involves building a platform for the integration of the various MBE toolchains that are commonly used in different stages of the development of a manufacturing system (using a framework called Arrowhead). The objective is to demonstrate how development and analysis of systems in MBE could become much easier if these tool chains could communicate with each other, when necessary, to overcome the limitations in the functionalities of the individual tools.

---

# Contents

|  |           |
|--|-----------|
| <b>I Translation</b>   | <b>4</b>  |
| <b>1 Introduction</b>  | <b>5</b>  |
| 1.1 Flexible Manufacturing Systems . . . . .                   | 5         |
| 1.2 Supervisory controllers . . . . .                          | 5         |
| 1.3 Model based engineering . . . . .                          | 6         |
| 1.4 Introduction to CIF . . . . .                              | 6         |
| 1.5 Introduction to LSAT . . . . .                             | 7         |
| 1.6 Research motivation . . . . .                              | 10        |
| 1.7 Problem definition . . . . .                               | 10        |
| 1.8 Preliminary research . . . . .                             | 10        |
| <b>2 Implementation</b>  | <b>12</b> |
| 2.1 Method I . . . . .   | 12        |
| 2.2 Method II . . . . .  | 16        |
| 2.3 Method III . . . . .                                       | 18        |
| 2.4 Method IV . . . . .  | 20        |
| 2.5 Method V . . . . .   | 22        |
| <b>3 Simulation and Results</b>                                | <b>24</b> |
| 3.1 Setup . . . . .  | 24        |
| 3.2 Results . . . . .  | 25        |
| 3.2.1 Number of sequences from <i>Seq</i> checked . . . . .    | 25        |
| 3.2.2 Time . . . . .   | 26        |
| <b>4 Conclusion</b>  | <b>28</b> |
| <b>II Tool chain implementation</b>                            | <b>29</b> |
| <b>5 Introduction</b>  | <b>30</b> |
| <b>6 Arrowhead Framework</b>                                   | <b>31</b> |
| <b>7 Implementation</b>  | <b>32</b> |
| <b>8 Conclusion</b>  | <b>32</b> |
| <b>References</b>  | <b>33</b> |
| <b>Acknowledgement</b>   | <b>35</b> |
| <b>Appendices</b>  | <b>36</b> |
| <b>A LSAT code: Machine specification for Twilight system</b>  | <b>36</b> |
| <b>B LSAT code: Settings specification for Twilight system</b> | <b>39</b> |
| <b>C LSAT code: Activity specification for Twilight system</b> | <b>42</b> |

---

|   |           |
|---|-----------|
| <b>D CIF code: Requirements</b>                       | <b>45</b> |
| <b>E Python code: Method I</b>                        | <b>46</b> |
| <b>F Python code: Method II</b>                       | <b>53</b> |
| <b>G Python code: Method III</b>                      | <b>61</b> |
| <b>H Python code: Method IV</b>                       | <b>71</b> |
| <b>I Python code: Method V</b>                        | <b>80</b> |
| <b>J Auxiliary code: subseqchecker</b>                | <b>90</b> |
| <b>K Auxiliary code: ConvertToAutomata</b>            | <b>93</b> |
| <b>L Declaration: TU/e Code of Scientific Conduct</b> | <b>97</b> |

---

**Part I**

**Translation**

---

# 1 Introduction

The onset of Industrial Revolution resulted in massive changes in the manufacturing sector. A sector that was previously dependent on producing most items manually shifted to the use of dedicated machinery, resulting in a leap in production volumes throughout the sector. The next big leap in this sector came around the start of the 20th century, when singular machines performing specific operations of a manufacturing process were integrated to form a manufacturing system. Consequently, streamlined end-to-end processes coupled with the advent of advanced electronic controllers resulted in even higher volumes and lower costs.

With time and change in market scenario, there came increased demand for product variety. Along with that came demands for manufacturing systems that could handle production of multiple varieties of product (or *recipes* as it is called in the manufacturing sector) while still retaining a high production capacity. However, the challenge with meeting these two demands in general is that delivering one demand usually requires a compromise of the other. On the one hand, lower variability in product types requires manufacturing systems tailored to produce a specific type of product with minimal stoppages, which result in increased production capacities. On the other hand, higher variability in product types requires manufacturing systems tailored to adapt to different product recipes using the same set of components.

## 1.1 Flexible Manufacturing Systems

To tackle these challenges and incorporate the demands of dynamic manufacturing environments, there has been a rapid rise in the need to develop flexible manufacturing systems (FMS)[1] after the turn of the century. In such systems, a high production rate along with a high degree of product variety is achieved through the use of versatile machinery and controllers that can quickly adapt to different recipes. Semiconductor manufacturing and automotive industries are examples where FMS are typically used.

Optimization of production capacity is key regarding an FMS. Generally, there are different products that may require different production processes and hence different production times. As a result, there is a necessity of meticulous scheduling and proper assignment of products to processes.

## 1.2 Supervisory controllers

As explained in [2] a supervisory controller ensures a system functions as desired by executing only the allowed sequence of tasks. The task sequences can be both within a component of the system or depend on multiple components as well. In general, a supervisory controller exercises control over the entire system by using information from its constituent parts.

The function of a supervisory controller (or *supervisor* in short and henceforth referred to as such) in the context of an FMS includes scheduling of operations. Generally, there is a fixed order of operations needed to be executed on a product and this is ensured by the supervisor. Hence, the productivity of a system depends not only on

---

the capabilities of the machines that execute the task but also on the supervisors that guide them.

However, to build a controller for a system, the system and its underlying processes need to be understood first before a supervisor can be synthesised, ultimately leading to such complex systems being realized physically. This is achieved by the application of model based engineering.

### 1.3 Model based engineering

Model based engineering (MBE) is an approach of representing a system through the usage of models [3]. Once a model representing the system has been built, it can be used for several purposes such as analysis, design of features, verification etc. The concept of MBE also naturally extends to the design and exploration of supervisors for a given system.

There are several tools that can be used for MBE and supervisor synthesis. However, in this thesis, the two tools that are dealt with specifically are mentioned below.

- Compositional Interchange Format (CIF)
- Logistics Specification and Analysis Tool (LSAT)

### 1.4 Introduction to CIF

Developed by researchers at Eindhoven University of Technology, CIF [4] is a tool which is a part of the Eclipse Supervisory Control Engineering Toolkit (Eclipse ESCET™)<sup>1</sup>. It is based on the principles of Supervisory Control Theory [5],[6] that was developed to integrate the process of MBE and supervisor synthesis. In CIF, a system is described by means of automata. Each automaton describes a part of the system. In its most basic form, an automaton is consisted of locations that describe the possible states that the sub-system might achieve. Initial locations indicate the state the sub-system starts in. Marked locations are used to describe states which are considered to be stable by the modeler of the system.

Events are used to model the dynamics of the subsystem by describing the transitions between the locations. Controllable events are used to describe events that, if necessary, can be disabled by a supervisor. Uncontrollable events are those which cannot be prevented from happening.

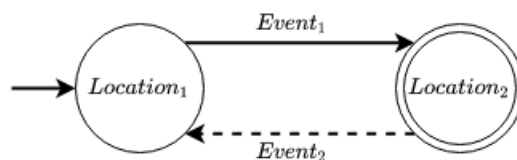


Figure 1: Automata example

---

<sup>1</sup>The ESCET toolset and documentation is open source and freely available at <https://www.eclipse.org/escet/>. 'Eclipse', 'Eclipse ESCET' and 'ESCET' are trademarks of Eclipse Foundation, Inc.



---

An example of an automaton is provided in Fig. 1, which describes a system that starts at the initial location  $Location_1$  (indicated by the dangling arrow) and transitions to marked location  $Location_2$  (indicated by double circles) if controllable event  $Event_1$  (indicated by the solid arrow) occurs. The system moves back to state  $Location_1$  when uncontrollable event  $Event_2$  (indicated by the dashed arrow) occurs.

In a similar vein to classical control theory, a *plant* in CIF terminology is used to refer to automata that describe the uncontrolled system behaviour. In addition, *requirements* that the system needs to fulfil, like certain events being possible only after others, can be specified. Using the plant and the requirements, a *supervisor* (which is analogous to a controller in classical control theory terms) for the system can be synthesised, which is also in the form of automata.

The reader is referred to [5],[6] for an in depth explanation of the principles behind the synthesis procedure. Two concepts, as explained briefly below, form the basis of supervisory synthesis.

- **Non-blockingness:** An automaton is deemed to be non-blocking if from any of its reachable states a sequence of events can occur which ultimately lead to a marked location. As explained previously, marked locations denote stable states. Hence, the concept of non-blockingness denotes the possibility of the system described by the automata to attain stability.
- **Controllability:** As explained previously, controllable events are events which can be disabled by a supervisor and uncontrollable events are the ones that cannot be prevented by a supervisor from occurring.

Given a plant and a set of requirements, the synthesis procedure then comes down to building an automaton that represents the combined system behaviour, and disabling the controllable events that either result in blocking or lead to a state with uncontrollable events which ultimately lead to blocking.

Alongside synthesis, CIF also has several other functionalities for specification and exploration of systems.

## 1.5 Introduction to LSAT

LSAT [7] is a modeling language set in the mould of MBE tools, typically used in the design of FMS. Developed jointly by ESI (TNO), ASML and Eindhoven University of Technology, the driving principle behind LSAT is design and exploration of supervisors that dictate the behaviour of an FMS. Along with a textual input interface, LSAT also allows a graphical interface using which the user can explore the behaviour of the system, perform analysis and implement optimization techniques.

The process of specifying the structure and behaviour of a system along with the supervisor that orchestrates the behaviour is made modular in LSAT by the usage of four integrated *domain specific languages* (DSL). Each DSL describes the system at a specific level of granularity, as a result of which a DSL of higher granularity has a dependency on a DSL of lower granularity. The DSLs and the aspect of a system described by each are listed below. The dependencies between the DSLs have also been mapped in Fig. 2.

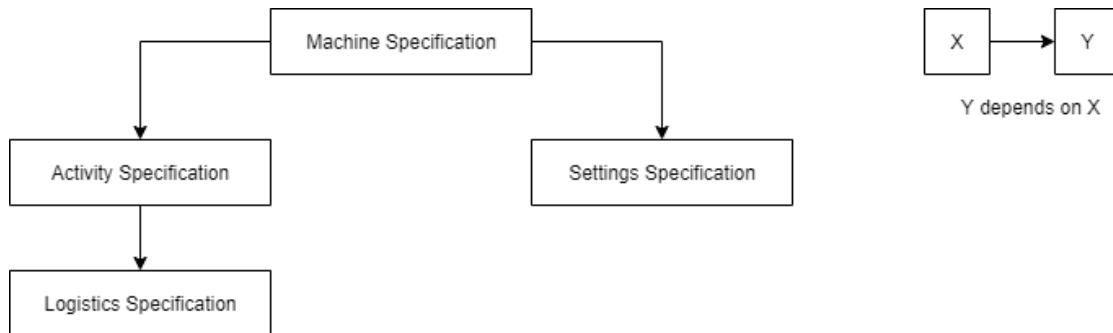


Figure 2: Dependencies between DSLs

- **Machine specification:** The fundamental language in the DSL hierarchy. This is used to describe the components of the system that can perform a pre-defined set of tasks. Such components are termed *resources*. A resource can be further broken down into sub components, called *peripherals*, that work in sync to perform the task asked of the resource.
- **Settings specification:** This is used to describe the physical settings of the peripherals defined in Machine specification. Physical settings include coordinates of allowed movement, motion profiles, etc.
- **Activity specification:** Using this language *activities* possible in a system are established. An activity is defined as a deterministic system operation consisting of *actions* that need to be performed in a definite acyclic order. An action, in turn, is a task that can be performed by a peripheral of a given resource as defined in the machine specification.
- **Logistics or Dispatching specification:** The final level in the DSL hierarchy, this is where the product flow in the system is established. The logistics specification defines the supervisor of the system in the form of a sequence of activities to be executed. A different sequence of activities implies a different product flow.

The Twilight system [8] shown in Fig. 3 is used as an illustration to explain the DSLs. It is a hypothetical system in which balls are processed according to a given recipe. The system is a simplified representation of a lithography scanner [9].

In the system, two robots move on a rail to transport balls; the Load Robot (LR) present on the left side of the rail picks unprocessed balls from the input (IN) and places the balls into the conditioning area (COND) for conditioning of the ball and the drill (DRILL) for drilling holes. A ball is considered processed if both conditioning and drilling operations are performed on it. Similarly, the unload robot picks up processed or semi-processed balls from COND and DRILL and places it in the output (OUT). The two robots each contain a clamp (CL) to pick up and hold a ball, an X-motor (X) that enables movement along the rail, and a Y-motor (Y) to move the clamp up and down. To limit the possibility that the two robots don't collide, a collision area (CA) has been defined, to prevent both robots from occupying the same position at any given moment. Additionally, the conditioner has a heater (H) to heat the ball while the DRILL has a Z-motor (Z) to move the drill bit up and down.

For the Twilight system, examples of certain aspects of the system specified using each DSL is given below.

- **Machine specification:**

- **Resources:** IN, LR, COND, DRILL etc.
- **Peripherals:** For LR and UR the peripherals are CL, X, Y etc. For DRILL, the peripherals are Z and so on.

- **Settings specification:** The acceleration profile of Z of DRILL, velocity profile of X of LR/ UR and so on.

- **Activity specification:** Actions move Y of LR down, turn CL on and finally move Y of LR back up together constitute the activity *pick product from IN* of LR and so on.

- **Logistics specification:** Activities of LR: pick product from IN, put product on COND, pick product from COND, put product on DRILL, and activities of UR: pick product from DRILL, put product in OUT can constitute an activity sequence.

LSAT also provides several tools for efficient analysis of systems. One such tool of particular significance in this research is that of makespan optimization. In this process however, supervisor for the system is not defined through the logistics specification directly. Instead, the supervisor is supplied via CIF in the form of automata with edges as activities as defined in the activity language.

The supervisor automata in turn are synthesised first by defining automata that allow

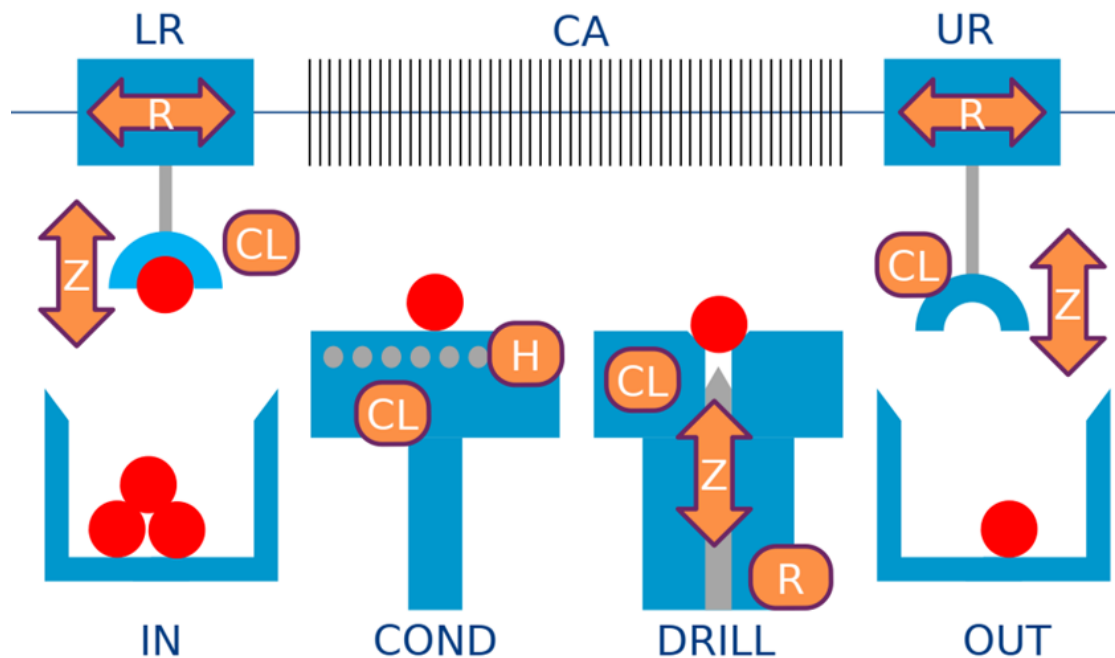


Figure 3: Twilight system. Adapted from [8]

---

all possible sequence of activities. Then, requirements are added to define the activity sequences allowed by the system. As a result, the synthesised supervisor contains all activity sequences allowed by the system. Once this is fed to LSAT, the activity sequence that provides the highest makespan is determined.

## **1.6 Research motivation**

As mentioned previously, at the moment, the supervisor supplied to LSAT for makespan optimization defines the activity sequences allowed by the system, which is synthesized by taking into account only the activity level requirements. There is no functionality that allows users to define requirements explicitly at action level. However, there could be situations in an actual manufacturing system where action level requirements have to be factored in. In such cases, having the functionality to incorporate such requirements enables a higher degree of control over the system. Hence, the objective of this research is:

*To find ways to develop the supervisor that reflect user defined action level requirements as well as activity level requirements.*

To illustrate, consider the following example from the Twilight system. As mentioned, the resource collision area is defined to prevent collision between the two robots, LR and UR. How this is achieved is each robot claims the collision area to perform any activity related to conditioning or drilling, and releases it on completion of the activity, which makes it available for the other robot to claim. Theoretically, if one robot moves fast enough (or the other too slow), it is possible that it claims the collision area immediately after its release, moves fast, and collides with the other robot. However, with action level requirements, the possibility of such occurrences can be eliminated by having requirements, for example, that prevent the two robots from occupying the same state.

## **1.7 Problem definition**

To achieve the aforementioned research objective, the broad approach adopted in this research is to find activity sequences that can fulfil the specified action and activity level requirements.

This work builds upon the groundwork laid in [10], wherein a method has been developed to represent the sequences along with activity and machine level specification in automata form. *Adding the requirements the actions need to fulfil to this representation, methods are to be devised such that the user, on application of these methods, knows exactly the sequences that can fulfil the given requirements.*

Using this information, a supervisor can be synthesized to be used by LSAT.

## **1.8 Preliminary research**

As mentioned previously, this work builds upon the groundwork laid in [10], which describes a methodology to represent a system described in LSAT using automata. The definition of the system elements, namely, for resources, actions, activities, and sequences, used in this work is taken from [10].

---

Activity instantiation is an important concept with regards to this work. By definition, an *instance* of an activity in a sequence is the occurrence number of the activity in the given sequence. The instance number is usually denoted using superscript. To illustrate, consider the sequence  $\omega$  in which activities  $Act_A$ ,  $Act_B$ ,  $Act_A$  are executed sequentially. Then,  $\omega = Act_A^1; Act_B^1; Act_A^2$ . Here, the first and second instances of  $Act_A$  are denoted by  $Act_A^1$  and  $Act_A^2$  respectively. Since there is only one instance of  $Act_B$ , it is denoted by  $Act_B^1$ . [10] also explains how multiple instances of the same activity can be executed simultaneously.

Subsequently, the steps to represent the behaviour of the system using *activity*, *claiming* and *availability* automata are also detailed (It is to be noted that the automata defining the peripheral behaviour have not been considered as part of the system behaviour as part of this research. Only the impact of introducing dependencies between actions of different activities are focused on).

The activity automaton for an activity instance contains as edges the constituent actions of the activity while incorporating the dependencies between the actions as defined in the activity definition. The claiming automaton for a given resource defines the order in which a particular resource is claimed by different activities for a given activity sequence. Finally, the availability automata for a resource ensures that a resource can be claimed only when it is available or has been released by another resource. The activity, claiming and availability automata together describe the behavior of the system for a given sequence to which user-defined requirements are added for synthesis.

---

## 2 Implementation

In this section, a few methods are discussed to find activity sequences that can fulfil a set of given requirements.

The LSAT specification elements are defined as per [10]. Furthermore, the following points are assumptions and definitions relevant to the implementation of the defined methods.

- The *maximum* length of the possible activity sequences ( $\mathcal{L}$ ) is defined by the user. The length of a sequence ( $l$ ) is defined as the number of activity instances in the sequence.
- When defining the requirements between different actions, the instances of the activities constituting the actions are specified, i.e., action instances are specified.
- All action instances specified in the requirements are labelled as *important actions*. The constituent activity instances are labelled as *important activities*. Similarly, all action instances not specified in the requirements are labelled as *unimportant actions*. The constituent activity instances are labelled as *unimportant activities*.

### 2.1 Method I

Consider a set of  $m$  activities specified in the activity specification

$$Act = \{Act_A, Act_B, \dots\} \text{ such that } |Act| = m$$

Let  $Seq$  be the set containing all activity sequences of length  $1 \leq l \leq \mathcal{L}$  that can be generated from  $Act$ . An arbitrary sequence  $\omega$  is selected from  $Seq$ . The activity, claiming and availability automata for  $\omega$  are generated as described in [10].

However, all the event edges are made uncontrollable. Furthermore, the final state of the activity automata, the final state of the claiming automata and the unclaimed state of the availability automata are deemed as marked as these are the states that can be deemed to be stable.

For example, consider  $Act = \{Act_A, Act_B\}$  as shown in Fig. 4 and a sequence of  $l = 3$ :  $\omega = Act_A^1; Act_B^1; Act_A^2$ . Then, the automata shown in Fig 5. describes the original behaviour of the system (plant) to which requirements are to be added.

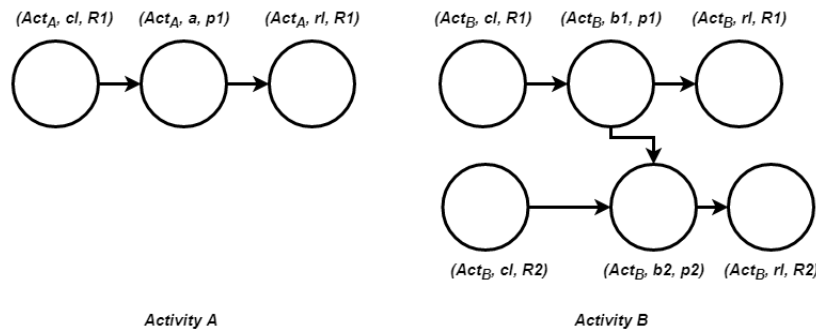


Figure 4: Example  $Act$

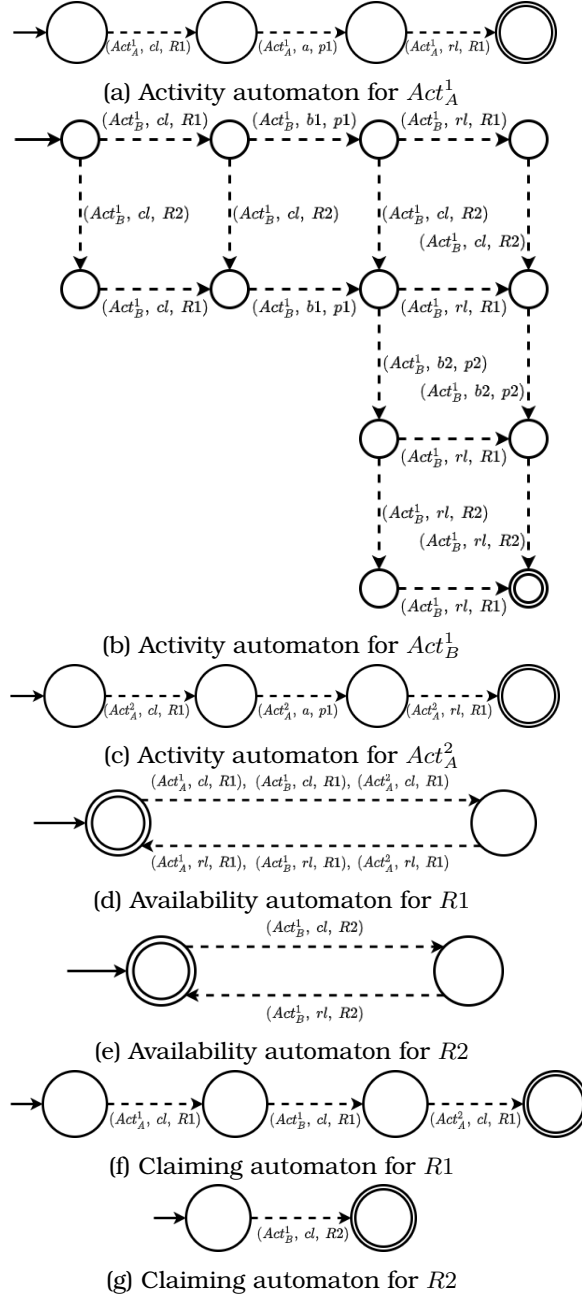


Figure 5: Example describing LSAT plant specifications in automata form

Now, if a set of requirements,  $\mathcal{R}$ , are added to the plant and synthesis is performed, two outcomes are possible as listed below:

1. **Empty supervisor:** This occurs if there is any behaviour in the combined system which does not conform to the given requirements ( $\mathcal{R}$ ). As all the edges are uncon-

---

trollable, removing of edges to meet the requirements are not allowed resulting in an empty supervisor

2. **Non-empty supervisor:** This occurs if there is no behaviour in the combined system which does not conform to the given requirements ( $\mathcal{R}$ ). The synthesised supervisor automaton is the plant itself as the edges remain unchanged due to their uncontrollable nature

Let the checking of the sequence using the aforementioned steps be denoted by the function  $check$  such that  $check(\omega|\mathcal{R}) = True$  if the given activity sequence  $\omega$  in presence of given requirements  $\mathcal{R}$  results in a non-empty supervisor and  $False$  otherwise.

Therefore, if  $check(\omega|\mathcal{R}) = True$ , it can be stated that the sequence  $\omega$ , under any conditions, fulfils the given requirements.

The aforementioned series of steps was to check if an arbitrary sequence of activities fulfils a given set of requirements. By extension, this can be repeated for all possible activity sequences in  $Seq$  to determine which sequences out of all possible sequences fulfil the requirements. Let  $Seq_{safe}$  denote the set containing all such sequences which fulfil the requirements. Then,  $Seq_{safe} \subseteq Seq$  such that  $\forall \omega \in Seq_{safe}, check(\omega|\mathcal{R}) = True$ .  $Seq_{safe}$  is populated by visiting all sequences in  $Seq$  one by one and checking if  $check(\omega|\mathcal{R}) = True$ .

Once all sequences are checked,  $Seq_{safe}$  can be used to synthesise a supervisor, which can then be used in LSAT. To illustrate, let  $Seq_{safe} = \{\omega_1, \omega_2\}$ , where  $\omega_1 = Act_A^1; Act_B^1$  and  $\omega_2 = Act_A^1; Act_C^1; Act_B^1$ . Then, one of the ways this can be used is by using a requirement automaton of the form shown in Fig. 6 while synthesizing the supervisor for LSAT. In general terms, this requirement automaton can be constructed by constructing an automaton in which by following the activities, as edges, of every sequence in  $Seq_{safe}$ , a marked state is reached. In other words, an automaton is made in which the sequences of activities in  $Seq_{safe}$  form its marked language.

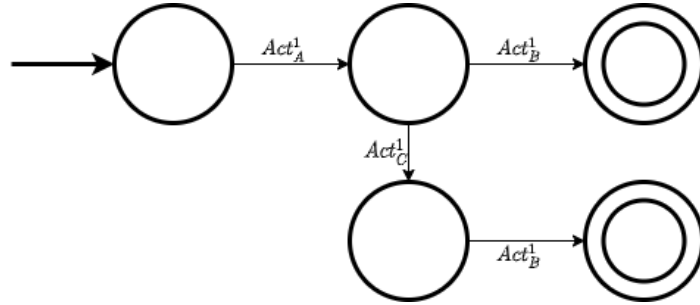


Figure 6: Example requirement automata to synthesize supervisor for LSAT

The complete process flowchart describing Method I is shown in Fig. 7.



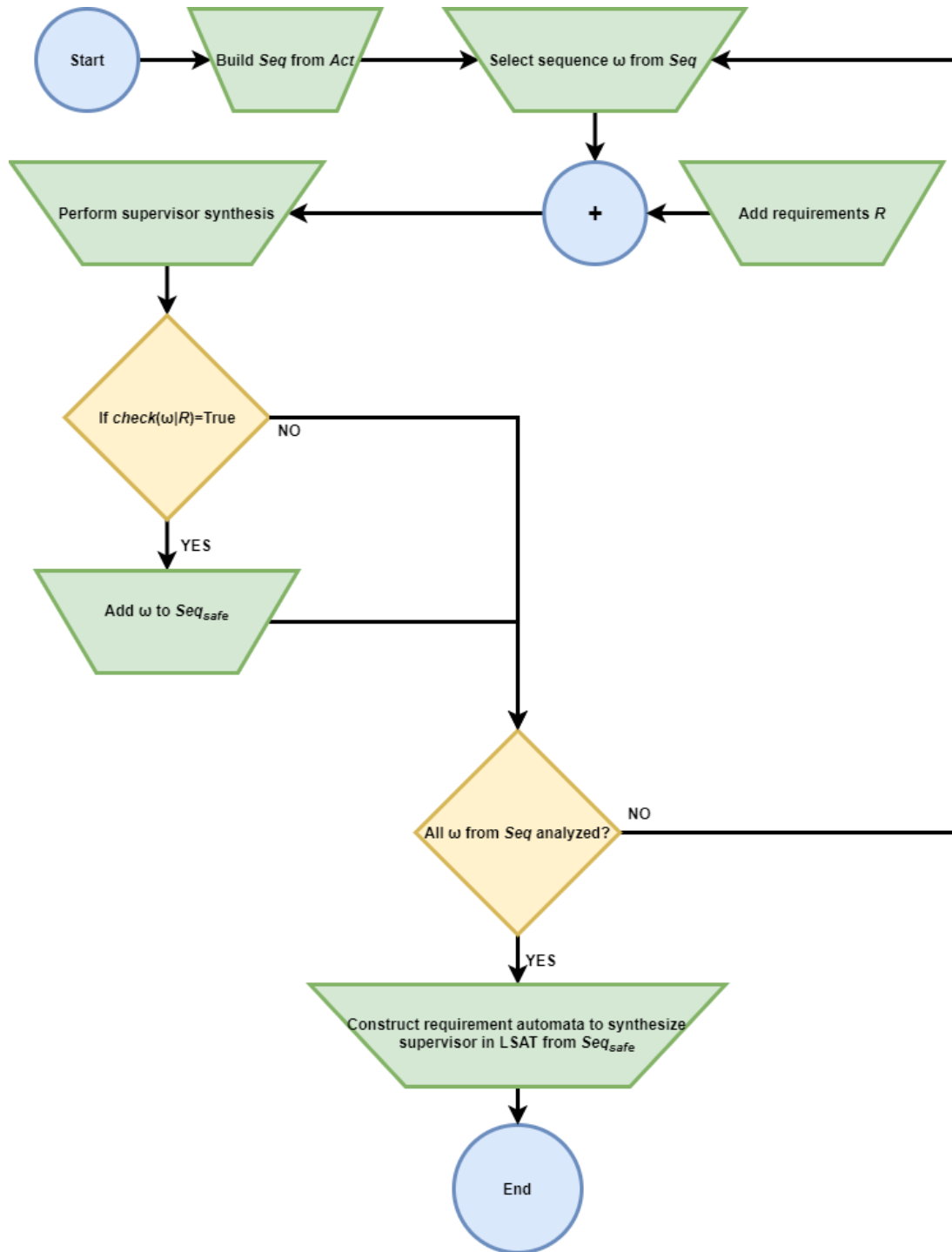


Figure 7: Process flowchart: Method I

---

## 2.2 Method II

This method is an extension of Method I. A few observations are stated prior to better understand the principle behind the method.

1. **Observation 1:** Given a sequence  $\omega$  such that  $check(\omega|\mathcal{R}) = True$ , then for any new sequence of the form  $\omega_{new} = \omega_1;\omega;\omega_2$ ,  $check(\omega_{new}|\mathcal{R}) = True$ , if  $\omega_1, \omega_2$  contain only unimportant activities. Similarly, given a sequence  $\omega$  such that  $check(\omega|\mathcal{R}) = False$ , then for any new sequence of the form  $\omega_{new} = \omega_1;\omega;\omega_2$ ,  $check(\omega_{new}|\mathcal{R}) = False$ , if  $\omega_1, \omega_2$  are empty sequences or contain only unimportant activities.

This is because  $\omega_1, \omega_2$  does not introduce edges or events that are contained in  $\mathcal{R}$ , as a result of which effectively the relationships between the events defined in  $\mathcal{R}$  remain the same as in  $\omega$

2. **Observation 2:** If  $check(\omega|\mathcal{R}) = True$ , then for any new sequence of the form  $\omega_{new} = \omega_a;\omega_0;\omega_b$ ,  $check(\omega_{new}|\mathcal{R}) = True$  if  $\omega_0$  is an arbitrary sequence containing only unimportant activities and  $\omega = \omega_a;\omega_b$ . This is in addition to Observation 1.

This can be explained by analyzing the DAG of the sequence  $\omega$ . The DAG of a sequence implies a DAG which contains the actions of the activities to be performed in sequential order. To satisfy the requirements defined in  $\mathcal{R}$ , the nodes of actions mentioned in  $\mathcal{R}$  have to be reachable from each other in a certain order. If they are not reachable implies the actions can occur concurrently. Introduction of activities that do not contain any action nodes present in  $\mathcal{R}$ , does not impact the reachability or ordering of the nodes present previously in  $\omega$ , thereby still satisfying the requirements.

In this method, the modification done with Method I is that not all sequences in  $Seq$  are checked individually. If a sequence fulfils the conditions mentioned under Observation 1 or Observation 2, then the explicit *check* for that particular sequence is not performed. Additionally, the sequences that fulfil the conditions under Observations 1 and 2 if  $check(\omega|\mathcal{R}) = True$  are appended to the  $Seq_{safe}$  set.

The motive behind the inclusion of the aforementioned steps is that it would lead to reduced computational effort and time as performing synthesis in addition to building the activity, claiming and availability automata is a computationally challenging procedure. In comparison, simply checking if a sequence fulfils the conditions mentioned under Observations 1 or 2 is a much simpler task and hence computationally less demanding.

The complete process flowchart describing Method II is shown in Fig. 8.

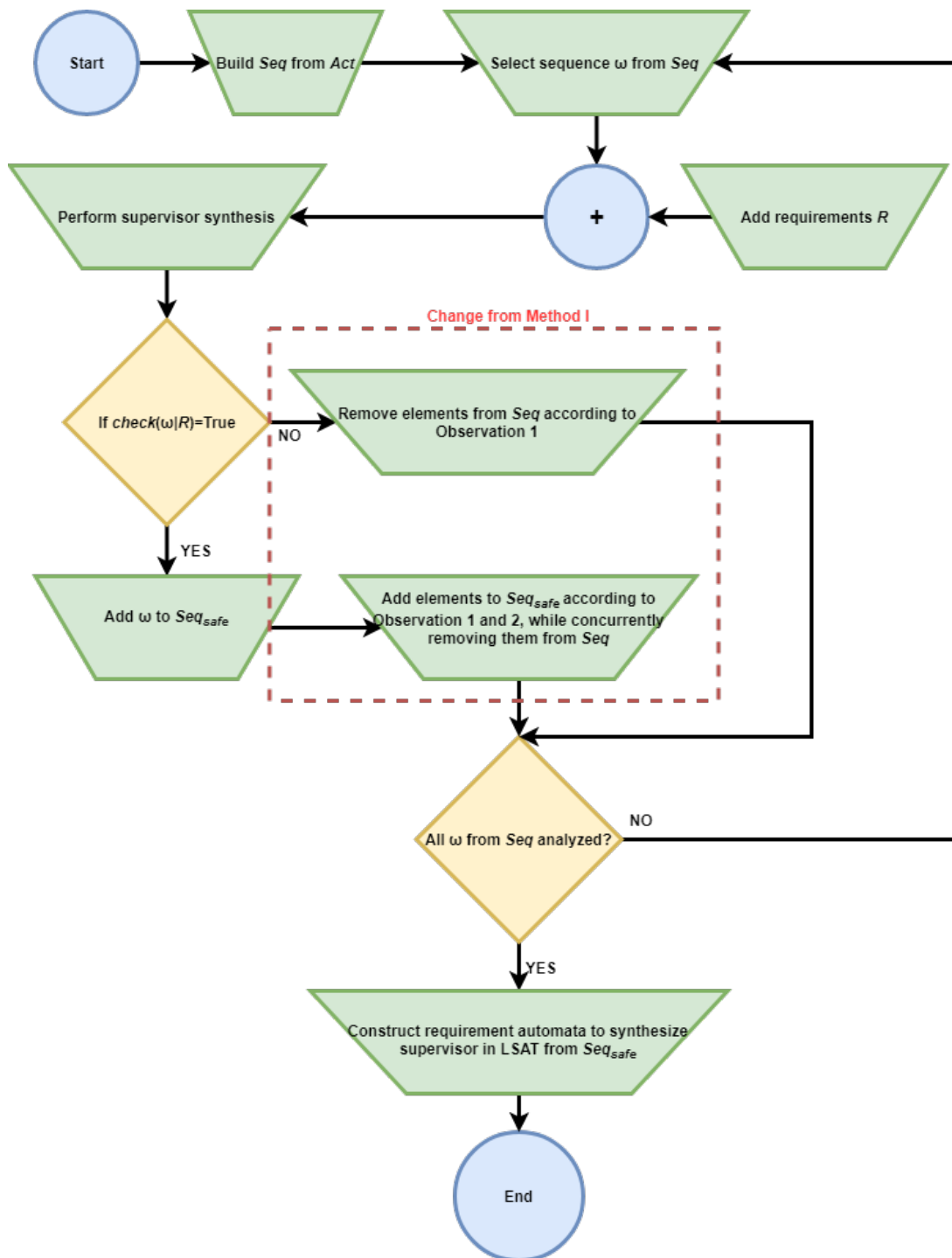


Figure 8: Process flowchart: Method II

---

## 2.3 Method III

This method is an extension of the previously described Method II and tries to reduce the computational effort and time to a higher extent. The technique followed to achieve that is *abstraction* of the activity automata that are generated when performing *check* operation for a given  $\omega$  and  $\mathcal{R}$ .

In general, abstraction is a process through which only relevant information (like states or events for an automaton) is used for computational purposes so that the computation load is reduced. In this case, the automata that are abstracted are the activity automata. The way the abstraction is performed is that for a given DAG of an activity in a sequence  $\omega$ , all nodes except claim, release and nodes of important actions are removed from the DAG. At the same time, whenever a node is removed, the predecessor nodes of the removed node are connected to the successor nodes. An example to illustrate the step is given in Fig. 9.



Figure 9: Illustration of abstraction of DAG of an activity

When trying to check if a sequence of activities fulfils a given requirement, what is checked in essence is only the relationship between the actions mentioned in the requirement i.e., the important actions. Therefore, the nodes of all other actions can be removed. However, claim/ release action nodes cannot be removed primarily for the two reasons mentioned below.

1. Claim actions in the claiming automata are used to describe the sequence of activities in a given sequence. Hence, removal of claim nodes will result in loss of this information while forming the claiming automata.
2. Claim/ release actions together are used in availability automata to describe when a resource is available to be used in an activity in a sequence. Removal of claim/ release nodes will therefore result in loss of this information while formation of the availability automata.

To illustrate, if an activity instance has unimportant actions only, then removal of the claim and release action nodes would result in an empty automaton for the particular activity, which would then imply that the activity is not part of the sequence. This would result in contradictions and consequently incorrect results.

This abstraction process works as all the necessary information, which is this case is the relationship between the important actions, is preserved even after removal process. It is to be noted that the supervisor generated while using  $check(\omega|\mathcal{R})$  for Method III is different from Methods I and II. However, the basic essence of these methods is to only check if a supervisor is possible or not. Hence, the composition of the supervisor is not as relevant.

Apart from the addition of the aforementioned steps of automata abstraction for the activities in a given sequence, the rest of the steps followed are the same as Method II. The complete process flowchart describing Method III is shown in Fig. 10.

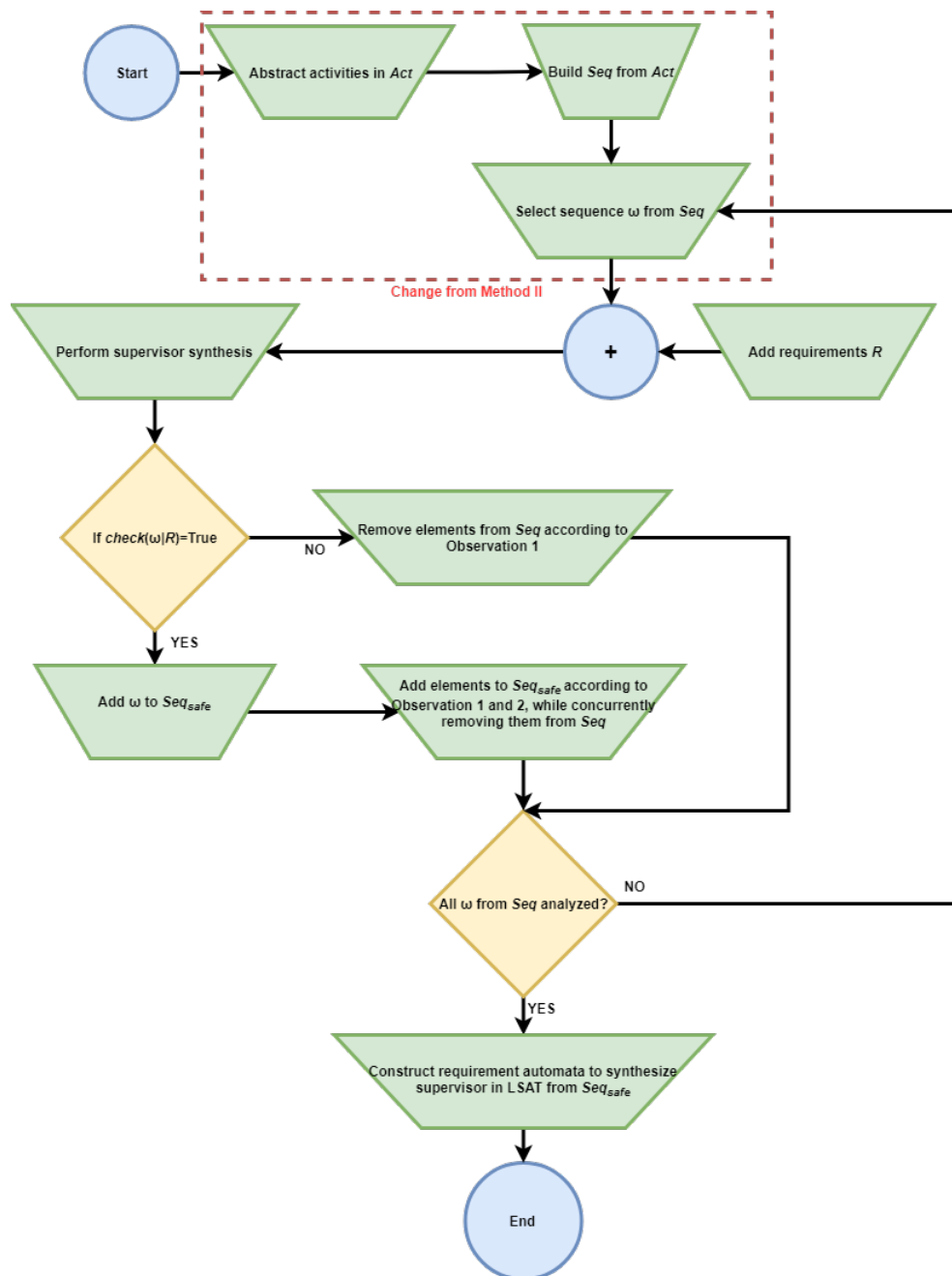


Figure 10: Process flowchart: Method III

## 2.4 Method IV

This method uses the principles of abstraction similar to the one used in Method III. However, this method deviates slightly from the step of setting up of activity, claiming and availability automata used thus far in Methods I, II, and III.

Instead of making activity automata for each constituent activity of a sequence  $\omega$  and then using the claim automata to describe the sequencing of the activities in the sequence, the combined DAG of the sequence  $\omega$  is made as described in [11], hereafter referred to as *sequence-DAG*. It is to be noted that the sequence-DAG describes all existing relationships between actions of the constituent activities of  $\omega$ . Following this, the sequence-DAG is abstracted in a manner similar to Method III, by removing all action nodes except nodes of important actions, while linking the edges from the predecessors to the successors of the removed nodes simultaneously. It is to be noted that the claim/release nodes can also be removed (unlike Method III) as they are now not explicitly necessary to describe the sequence of activities in a given sequence or indicate the availability of resources to be used in activities.

An example to illustrate the step is shown in Fig. 11, where  $\omega = Act_A^1; Act_B^1; Act_A^2$ , and  $act_A, Act_B$  is as given in Method I. Consider  $(Act_A^1, a, p1)$ ,  $(Act_A^2, a, p1)$ ,  $(Act_B^1, b2, p2)$  as important actions.

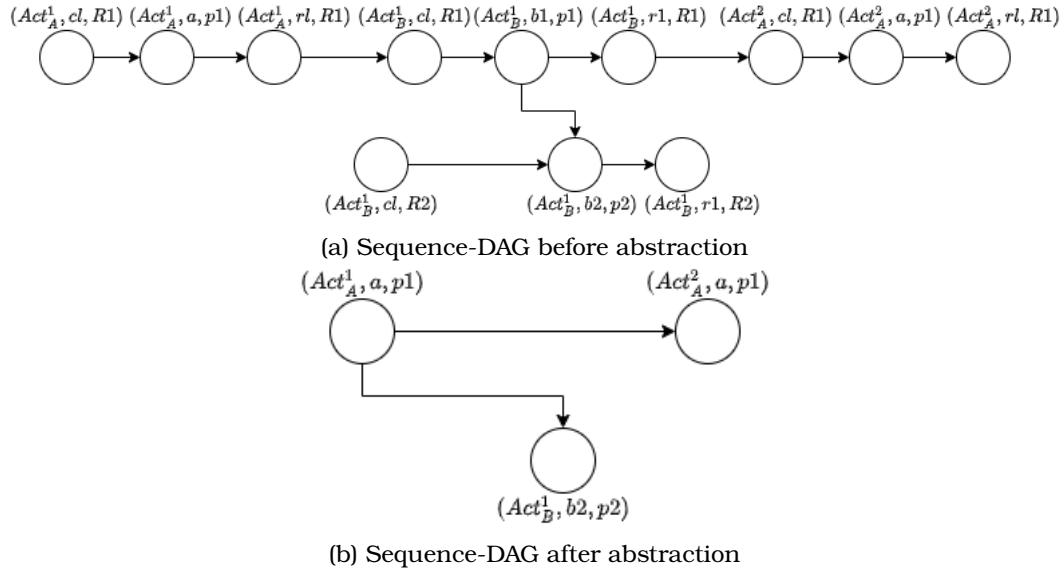


Figure 11: Illustration of abstraction of sequence-DAG

Once the abstraction of the sequence-DAG is complete, a *sequence automaton* is extracted by following the methodology described in [10] to extract the activity automaton. The sequence automaton can then be used along with requirements,  $\mathcal{R}$ , to synthesize a supervisor and determine whether  $check(\omega|\mathcal{R}) = True$ .

Apart from the aforementioned steps, the rest of the methodology is similar to Methods II and III. The process flowchart describing Method IV is shown in Fig. 12.

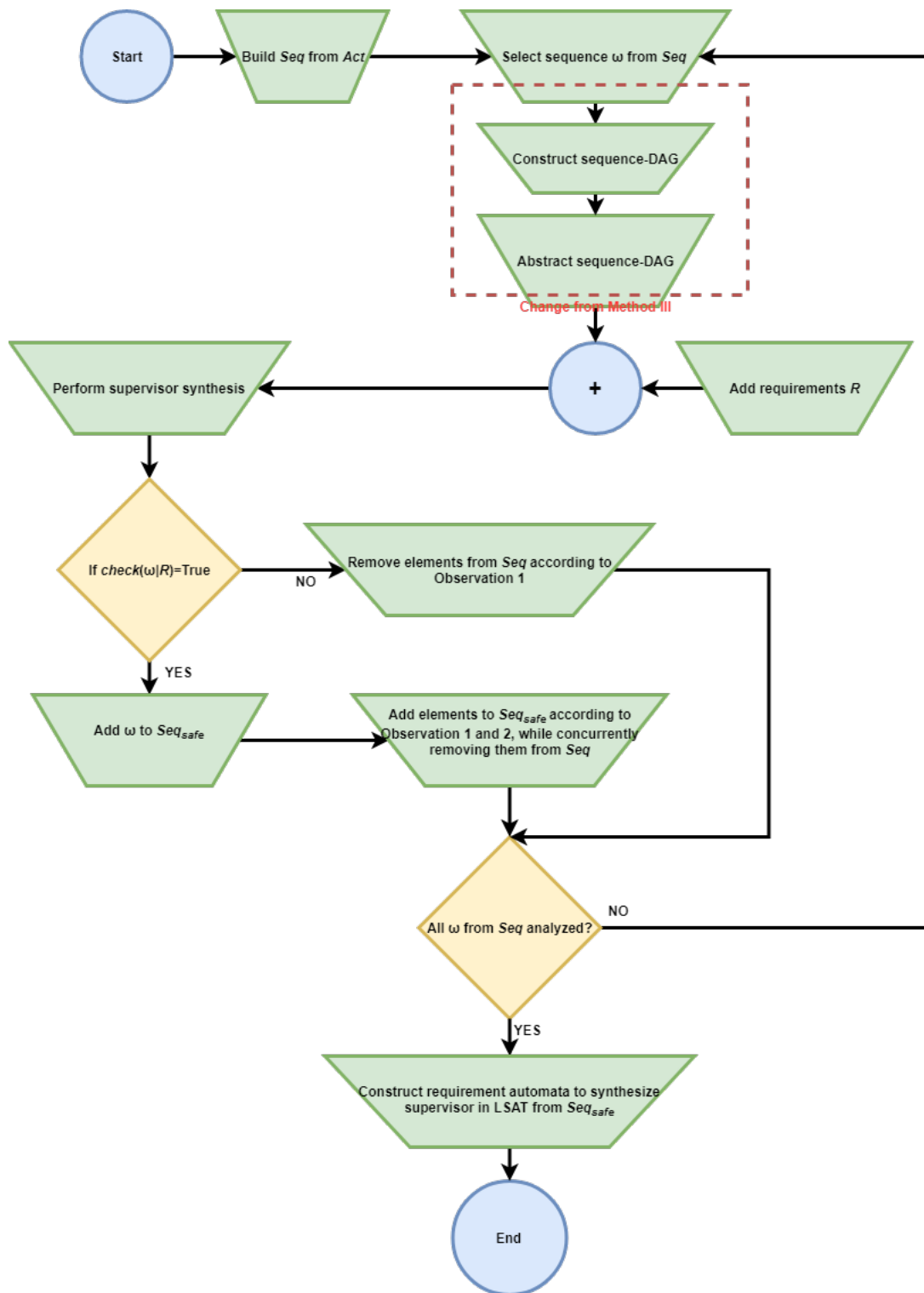


Figure 12: Process flowchart: Method IV

---

## 2.5 Method V

This method is an extension applicable to all methods discussed above but in this research it is applied as an extension to Method IV.

In all methods described, the aim has been to reduce the computational time and effort required gradually either by reducing the number of sequences (like Method II) or through abstraction (like Methods III and IV). In this method, the former approach is followed by trying to reduce the initial set that holds all possible sequences to be visited  $Seq$ .

This is achieved by filtering the sequences allowed by the supervisor synthesized from a plant that allows all possible sequence and activity level requirements. Let the set containing the possible sequences be  $Seq_{new}$ .

To illustrate, if it is known before hand that due to constraints (e.g., floor plan) the only possible sequence of activities are the ones where the first instance of  $Act_B$  can occur only after the first instance of  $Act_A$ , then  $Seq_{new} = Seq \setminus \{\Omega_{np}\}$ , where  $\Omega_{np}$  is the set containing all sequences  $\omega_{np}$  such that  $\omega_{np} = \omega_1; Act_B^1; \omega_2; Act_A^1; \omega_3$ , and  $\omega_{1,2,3}$  are arbitrary sequences.

The idea behind this method is self-explanatory. If  $Seq_{new}$  contains lesser number of sequences as compared to  $Seq$ , it automatically reduces the number of sequences to be analyzed. It is to be noted that if there are no dependencies stated between the activities, then  $Seq_{new} = Seq$

The complete process flowchart describing Method V is shown in Fig. 13.



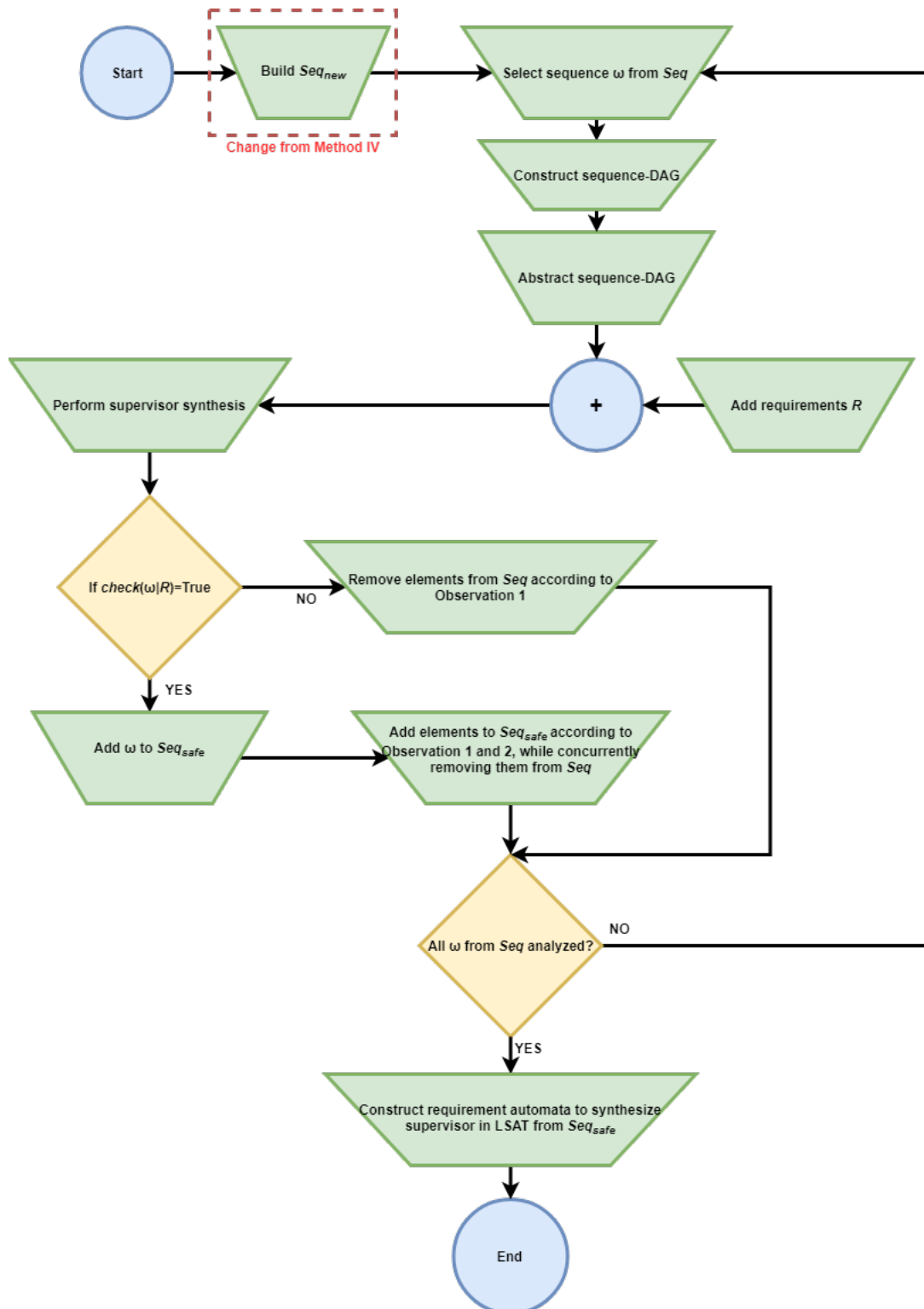


Figure 13: Process flowchart: Method V

---

## 3 Simulation and Results

In this section, the performances of the methods described are compared by applying them on a given system and requirements.

### 3.1 Setup

The example system used in this section is the Twilight system described previously. However, for the purpose of illustration in this report, only the activities involving the LR are taken into consideration, i.e., the set *Act* contains the following elements:

- *LR.PickPrdFromInput*
- *LR.PutPrdOnCond*
- *LR.PutPrdOnDrill*
- *LR.PickPrdFromCond*
- *LR.PickPrdFromDrill*

Moreover, it is assumed that the ordering of the activities are not strict, i.e, unless specified in the form of requirements (as done for Method V), any activity can follow any other activity or itself while forming a sequence from the given set *Act*. The machine, settings and activity files of the Twilight system are provided in Appendix A, B, and C, respectively.

The following are the dependencies between action instances which are used in this illustration. In automata form, they are as shown in Fig. 14. The respective CIF file for the requirements is as shown in Appendix D.

1. The first instance of action *a3*: Conditioner.CL.clamp of activity *LR.PutProdOnCond* can be performed only after the first instance of action *a3*: move LoadRobot.XY to ABOVE\_IN with speed profile normal of *LR.PickPrdFrmInput* has been performed
2. The second instance of action *a1*: move LoadRobot.XY to ABOVE\_COND with speed profile normal of *LR.PutPrdOnCond* can only be performed if at least one instance of action *a5*: Drill.CL.clamp of *LR.PutPrdOnDrill* has already been performed

For Method V, the requirement for the activities in the sequences is as shown in Fig. 15 and is mentioned below:

- The first instance of *LR.PickPrdFromCond* can be performed only when an instance of *LR.PutPrdOnCond* has occurred, which in turn can be performed only when an instance of *LR.PickPrdFromInput* has been undertaken

The codes for the various methods are detailed in Appendix E-I. All snippets have been commented for clarity.

The metrics used for evaluating the methods are described below:

- **Number of sequences from *Seq* checked:** As performing a *check* for a sequence requires supervisory synthesis which is a computationally challenging procedure, the lower the number of *check* operations performed, the better the performance of the method.

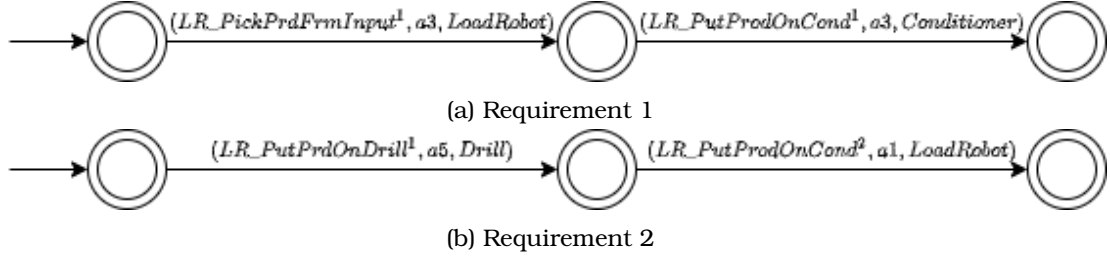


Figure 14: Automata describing the defined requirements between the action instances

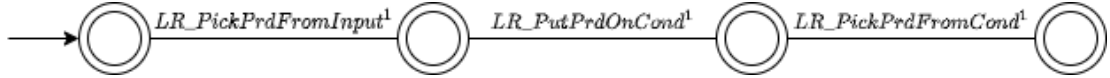


Figure 15: Automata describing the defined requirements between the action instances

- **Time:** A straightforward metric analyzing the time taken by the different methods. One of the factors this depends on is the number of *check* operations performed. However, abstraction has also been a technique applied to some of the methods. Therefore, it is expected that a method implementing a more effective abstraction technique will require less time to complete overall.

The performance of the discussed methods were evaluated by gradually increasing the max length of the activity sequences,  $\mathcal{L}$ , and comparing the aforementioned metrics in each of the case. The results are produced in the subsequent subsection.

## 3.2 Results

### 3.2.1 Number of sequences from $Seq$ checked

|            | Max length of sequence ( $\mathcal{L}$ ) |    |     |     |      |       |
|------------|--|----|-----|-----|------|-------|
|            | 1  | 2  | 3   | 4   | 5    | 6     |
| Method I   | 5  | 30 | 155 | 780 | 3905 | 19530 |
| Method II  | 5  | 12 | 33  | 122 | 532  | 2442  |
| Method III | 5  | 12 | 33  | 122 | 532  | 2442  |
| Method IV  | 5  | 12 | 33  | 122 | 532  | 2442  |
| Method V   | 3  | 6  | 10  | 19  | 49   | 186   |

Table 1: No. of sequences from  $Seq$  checked

As can be observed from Table 1, the number of sequences from  $Seq$  needed to be checked keeps increasing exponentially with the increase in max size of sequence when using Method I. This is because all sequences in  $Seq$  are checked, which in turn increases by a value of  $N^l$  with increase in the length of sequence from  $l - 1$  to  $l$ , where  $N = size(Act)$ . In the case of this example,  $N = 5$ . So, as the length of sequence increases from  $l = 1$  to  $l = 2$ , the number of sequences checked increases from 5 to  $5 + 5^2 = 30$ , and so on.

However, a drastic improvement in the results can be observed when using Methods II-IV, which, as discussed before, selectively checks sequences from  $Seq$ . As expected,

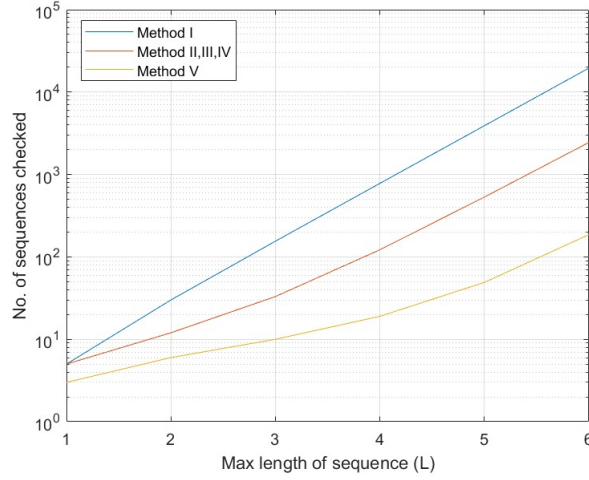


Figure 16: No. of sequences checked vs max length of Sequence ( $\mathcal{L}$ )

the number of sequences visited are the same for Methods II-IV as methods II and IV does not employ any new technique to reduce the number of sequences visited, but employs different abstraction techniques while checking a sequence. As can be seen from Fig. 16 however, the number of sequences checked still trend to be exponentially increasing but the rate is much lower as compared to Method I.

With the application of Method V, however, the results are significantly better, as  $Seq$  is reduced significantly by analyzing the supervisor synthesized from the requirements defined solely for the activities.

### 3.2.2 Time

|            | Max length of sequence ( $\mathcal{L}$ ) |      |       |       |        |         |
|------------|--|------|-------|-------|--------|---------|
|            | 1  | 2    | 3     | 4     | 5      | 6       |
| Method I   | 4.8                                      | 34.2 | 153.3 | 804.6 | 4352.3 | 22028.8 |
| Method II  | 5.1                                      | 11.9 | 32.3  | 126.5 | 600.5  | 2636.5  |
| Method III | 4.9                                      | 10.9 | 31.2  | 121.1 | 567.9  | 2511.3  |
| Method IV  | 4.9                                      | 11.1 | 30.2  | 115.6 | 513.8  | 2345.5  |
| Method V   | 2.9                                      | 5.7  | 9.6   | 18.0  | 45.9   | 184.6   |

Table 2: Time (s)

As expected and can be observed from Table 2, with the increase in the number of checked sequences, the time taken to determine the sequences that fulfil the given requirements also increases. As can be seen from Fig. 17, the rate of increase for Methods I and II reflects the same observation.

Method III delivers better results as compared to Method II as the activity automata are abstracted to contain lesser number of states and hence, the time to perform supervisory synthesis for each sequence also reduces, which the reader might recall is necessary to

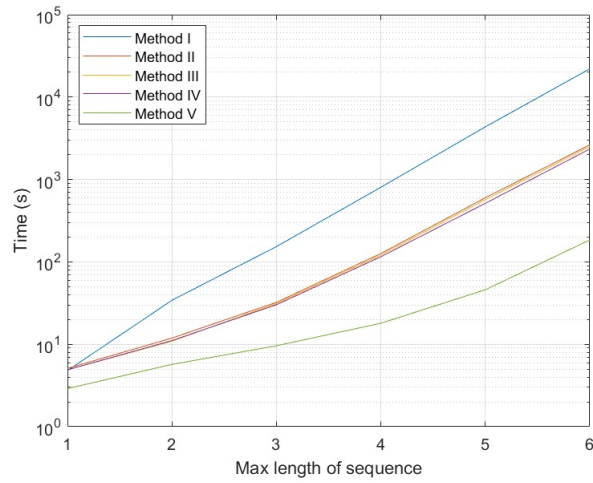


Figure 17: Time (s) vs max length of Sequence ( $\mathcal{L}$ )

perform  $check(\omega|\mathcal{R})$ . Furthermore, the difference in time taken increases as  $\mathcal{L}$  increases, as more activity automata are abstracted. Method IV improves further upon Method III as the abstraction is to a higher degree only containing important activities.

Method V, as explained, shows better results compared to the other methods, which can be attributed mainly to the reduction in the number of sequences in  $Seq$ .

---

## 4 Conclusion

Through this research different techniques were established to enable users of LSAT to model supervisors which not only capture the dependencies between activities but also the constituent actions of the activities, which offers a higher degree of control while designing systems. The different techniques were compared to establish which method would scale better while application in an industry setting and it can be concluded that Method V suits the best due to its better performance metrics compared to the other methods. Furthermore, it is to be noted that in all the methods discussed, memory constraint is not a major concern (and hence has not been treated as a metric for evaluation) as every sequence is checked individually. As a result, the size of the supervisors synthesized are also limited. Using abstraction reduces the size to a greater extent.

As the example showed, with the addition of correct requirements, the time taken for a supervisor synthesis can be reduced greatly. Taking into consideration the entirety of the Twilight system along with the correct and well-defined requirements, both at the activity and action level, synthesis of a supervisor using Method V is a feasible task. Consequently, Method V can be adopted for real life industrial scenarios given that the system is well understood and the dependencies between the various activities are taken into account as this reduces the initial search set of activity sequences greatly.

This work can be used as a building block for further work in this domain. A few suggested guidelines are mentioned below.

- In all the methods discussed, supervisory synthesis is performed to essentially check whether certain behaviour is always allowed in the system (as all edges are uncontrollable). Instead of supervisory synthesis, the concept of *model checking* could be explored as an approach to perform the same operation, which could potentially lead to better performing methods.
- Another step in improving upon this research work could be to develop techniques that allow a higher degree of control by incorporating requirements at the peripheral and resource level.
- Right now, the methods that were discussed all check sequences one by one to determine if a sequence fulfils given requirements. However, as can be seen from the results, the scalability of the approach is not very good, especially for sequences of larger length. In this regard, another approach that can be explored is first building a supervisor that contains all allowed behaviour and extracting the sequences by analyzing the supervisor.

---

**Part II**

# **Tool chain implementation**

## 5 Introduction

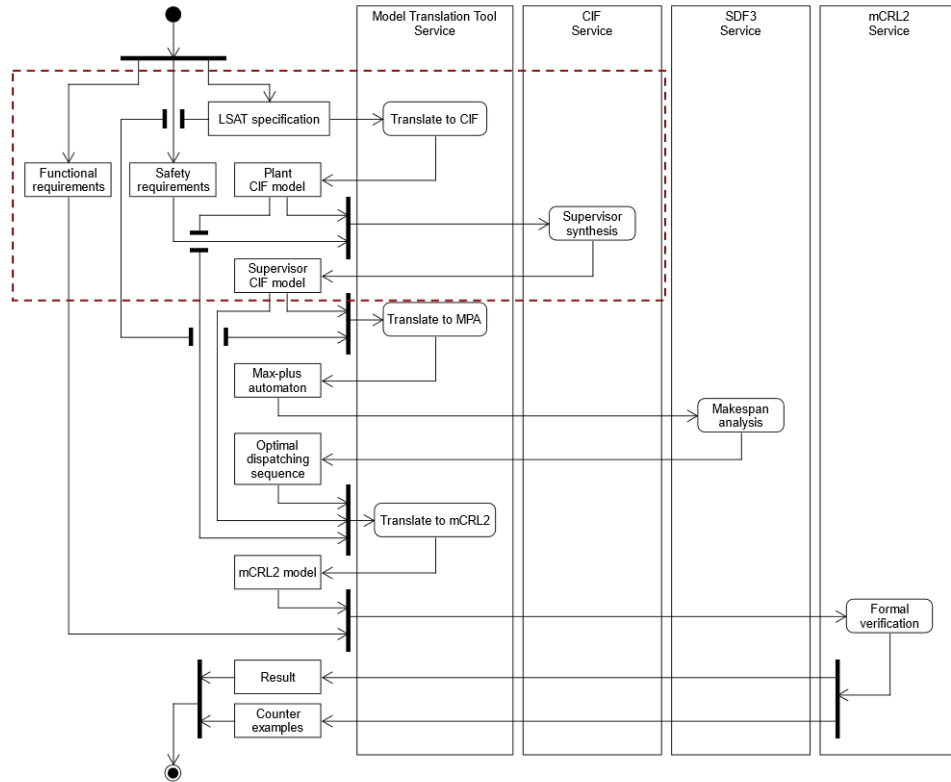


Figure 18: Arrowhead Framework example

Development of systems using MBE involves various stages, starting from design to verification and analysis. Naturally, different tools are needed at different stages of the process. A few tools used in industry are LSAT (for system specification), CIF (for supervisor synthesis), SDF3 [12] (to perform timing analysis), mCRL2 [13] (formal verification).

A typical toolchain schematic showing the various steps involved in the design of a system is shown in Fig. 18. These individual tools, however, have different specification, operate on different types of licenses, and have different purposes. Development and analysis of systems in MBE could become much easier if these tools could communicate with each other when necessary to overcome the limitations in the functionalities of the individual tools. Additionally, operations performed by one or more of these tools could be computationally challenging. In such cases, having a tool running on a powerful system to which other tools can communicate when necessary can prove useful.

The work described in the previous part of this report is the LSAT to CIF translation component of the toolchain. Once an effective tool has been built, integration of the tool within the toolchain needs to be performed; the process of which has been illustrated in this part.



---

For this purpose, this part of the research aims to use Arrowhead Framework 1 to set up a local cloud instance consisting of LSAT (service consumer), the developed model translator (service provider 1) and CIF (service provider 2) as clients with the framework orchestrating HTTP requests securely between the toolchains. In turn, this test case would aid in highlighting the ease of deployment of local clouds and IoT automation systems using Arrowhead. This has been highlighted in Fig. 18 .

## 6 Arrowhead Framework

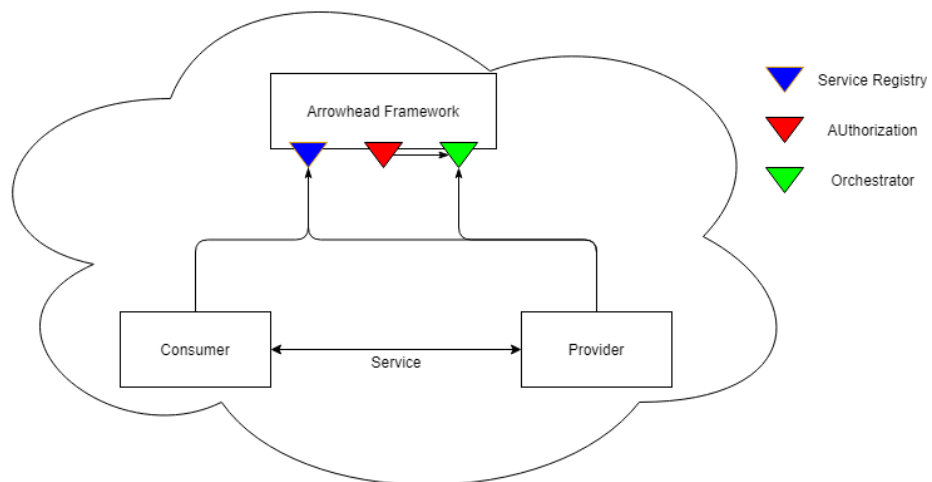


Figure 19: Arrowhead Framework example

The Arrowhead Framework [14] developed by the Eclipse foundation consists of tools that can be used for designing, implementing, and deploying Automation Systems compliant with Industry 4.0 and RAMI 4.0. The framework of Eclipse Arrowhead directs its users to adopt a common and unified approach in turn achieving high levels of interoperability. The approach taken is that IoT's are abstracted to services. This enables IoT interoperability in almost any IoT's. The automation is based on the concept of setting up of local automation clouds.

In its most simple form, a local cloud consists of a consumer and a provider of service with Arrowhead framework providing three core services: service registry, authorization, and orchestration as shown in Fig. 19. A service itself is realized in the form of HTTP request response cycles. The sequence of operations in a cloud is usually as follows:

1. The service provider and consumer system register the services provided by them in the cloud by sending a request to the service registry of the framework
2. Once the services are registered in the registry, authorization rules are set by the user to define specific provider services that can be used by the consumer

- 
3. Finally the orchestrator controls the actual consumption of the service by scanning the cloud for the service desired by the consumer according to the authorization rules set. Additionally, an orchestrator store of services can also be set up which tells the orchestrator the exact service needed by a consumer system.

The aforementioned is an instance with a single local cloud. Many other features may be added, such as multiple clouds, multiple providers-consumers, systems with subscribers and publishers, etc.

## **7 Implementation**

The following were the steps followed in the setting up of a local arrowhead cloud:

1. The LSAT to CIF translator was designed. In this case, the translator could build a plant flower automata in CIF containing all activities in the provided LSAT model.
2. The next step was to design generic wrappers for the services provided. A wrapper is a program generically written to communicate with the program actually performing the task
3. Finally, the provider wrappers were connected to the translator program and CIF synthesis executable as these were the two service providers for this illustration. A consumer wrapper was created to accept LSAT files and communicate with the two service providers depending upon the input of the user

Upon implementation, the user could select a LSAT file via the consumer interface to be sent to the server containing the provider and receive a translated CIF file. Furthermore, the CIF file could then be selected via the consumer interface along with user defined requirements and sent to the server containing the CIF synthesis executable and receive the synthesised supervisor CIF model.

## **8 Conclusion**

Through this exercise, the necessity of building a connected toolchain and the ease of development using a framework such as Arrowhead was demonstrated.

The toolchain is planned to be expanded in the future by incorporating other tools in the toolchain in the IoT cloud. In addition, a translator which incorporates the action level behaviour and requirements, as described in the previous part, can easily be incorporated into the toolchain.

---

## References

- [1] L.J. van der Sanden. “Performance analysis and optimization of supervisory controllers”. PhD thesis. Electrical Engineering, Nov. 2018. ISBN: 978-94-6380-057-0. URL: [https://pure.tue.nl/ws/portalfiles/portal/109096741/20181122\\_Sanden.pdf](https://pure.tue.nl/ws/portalfiles/portal/109096741/20181122_Sanden.pdf).
- [2] R.J.M. Theunissen. “Supervisory control in health care systems”. PhD thesis. Mechanical Engineering, 2015. ISBN: 978-90-386-3789-1. URL: <https://pure.tue.nl/ws/portalfiles/portal/3960230/786117.pdf>.
- [3] N. Shevchenko. *An Introduction to Model-Based Systems Engineering (MBSE)*. 2020. URL: <https://insights.sei.cmu.edu/blog/introduction-model-based-systems-engineering-mbse/>.
- [4] D.A. van Beek et al. “CIF 3 : model-based engineering of supervisory controllers”. In: *Tools and algorithms for the construction and analysis of systems*. Ed. by E. Abraham and K. Havelund. Lecture Notes in Computer Science. Germany: Springer, 2014, pp. 575–580. ISBN: 978-3-642-54861-1. DOI: 10.1007/978-3-642-54862-8\_48.
- [5] W. M. Wonham and P. J. Ramadge. “On the supremal controllable sublanguage of a given language”. In: *The 23rd IEEE Conference on Decision and Control*. 1984, pp. 1073–1080. DOI: 10.1109/CDC.1984.272178.
- [6] W.M. Wonham. “Supervisory Control of Discrete-Event Systems”. In: *Encyclopedia of Systems and Control*. Jan. 2013, pp. 1–10. ISBN: 978-1-4471-5102-9. DOI: 10.1007/978-1-4471-5102-9\_54-1.
- [7] L.J. van der Sanden and Y. Blankenstein et al. “LSAT: Specification and Analysis of Product Logistics in Flexible Manufacturing Systems”. In: *IEEE 17th International Conference on Automation Science and Engineering (2021)*. URL: <https://a.storyblok.com/f/74249/x/225d57f1c5/lsat-paper.pdf>.
- [8] L.J. van der Sanden et al. “Compositional specification of functionality and timing of manufacturing systems”. Netherlands. In: *Proceedings of the 2016 Forum on specification and Design Languages, FDL 2016, Bremen, Germany, September 14-16, 2016*. URL: <https://ecsi.org/fdl>.
- [9] L.J. van der Sanden et al. “Modular model-based supervisory controller design for wafer logistics in lithography machines”. In: *2015 ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. Sept. 2015, pp. 416–425. ISBN: 978-1-4673-6909-1. DOI: 10.1109/MODELS.2015.7338273.
- [10] S.B. Thuijsman and M.A. Reniers. “Conversion of LSAT behavioral specifications to automata”. In: *arXiv 2020* (Nov. 2020). URL: <https://arxiv.org/pdf/2011.03249.pdf>.
- [11] J.P. Nogueira Bastos. “Modular specification and design exploration for flexible manufacturing systems”. PhD thesis. Electrical Engineering, Dec. 2018. ISBN: 978-94-6380-091-4. URL: [https://pure.tue.nl/ws/portalfiles/portal/111655386/20181203\\_Bastos.pdf](https://pure.tue.nl/ws/portalfiles/portal/111655386/20181203_Bastos.pdf).

- 
- [12] S. Stuijk, M.C.W. Geilen, and T. Basten. “SDF3: SDF For Free.” In: *Proceedings of the 6th International Conference ACSD 2006*. Ed. by K.G.W. Goossens and L. Petrucci. United States: IEEE Computer Society, 2006, pp. 276–278. ISBN: 0-7695-2556-3. URL: [https://www.es.ele.tue.nl/sdf3/publications/acsd06\\_sdf3.pdf](https://www.es.ele.tue.nl/sdf3/publications/acsd06_sdf3.pdf).
- [13] J.F. Groote et al. “The mCRL2 toolset”. In: *Informal proceedings of the International Workshop on Advanced Software Development Tools and Techniques*. 2008, pp. 5–1/10. URL: <https://www.jeroenkeiren.nl/assets/publications/GKM+08.pdf>.
- [14] R. Rocha et al. “The Arrowhead Framework applied to energy management”. In: *2018 14th IEEE International Workshop on Factory Communication Systems (WFCS)* (2018), pp. 1–10. doi: 10.1109/WFCS.2018.8402357.

---

## **Acknowledgement**

Research leading to these results has received funding from the EU ECSEL Joint Undertaking under grant agreement n° 826452 (project Arrowhead Tools) and from the partners national programs/funding authorities.

---

# Appendices

## A LSAT code: Machine specification for Twilight system

```
1 Machine Twilight
2
3 PeripheralType Clamp {
4     Actions {
5         clamp
6         unclamp
7     }
8 }
9
10 PeripheralType XYMotor {
11     SetPoints {
12         X [m]
13         Y [m]
14     }
15     Axes {
16         X [m] moves X
17         Y [m] moves Y
18     }
19 }
20
21 PeripheralType DrillMotor {
22     Actions {
23         on
24         off
25     }
26 }
27
28 PeripheralType Drill {
29     SetPoints {
30         Z [m]
31     }
32     Axes {
33         Z [mm] moves Z
34     }
35     Conversion "Z=Z/1000"
36 }
37
38 PeripheralType Conditioner {
39     Actions {
40         condition
41     }
42 }
```

---

```

43
44 Resource Drill {
45     CL: Clamp
46     DL: DrillMotor
47     ZR: Drill {
48         SymbolicPositions {
49             UP
50             DOWN
51         }
52         Profiles (normal, slow)
53         Paths {
54             DOWN --> UP profile slow
55             UP --> DOWN profile normal
56         }
57     }
58 }
59
60 Resource Conditioner {
61     CL: Clamp
62     CD: Conditioner
63 }
64
65 Resource LoadRobot {
66     CL: Clamp
67     XY: XYMotor {
68         AxisPositions {
69             X (IN, COND, DRILL)
70             Y (ABOVE, AT)
71         }
72         SymbolicPositions {
73             ABOVE.IN (X.IN, Y.ABOVE)
74             ABOVE.COND (X.COND, Y.ABOVE)
75             ABOVE.DRILL (X.DRILL, Y.ABOVE)
76             AT.IN (X.IN, Y.AT)
77             AT.COND (X.COND, Y.AT)
78             AT.DRILL (X.DRILL, Y.AT)
79             OUT.DRILL (X.DRILL)
80         }
81         Profiles (normal)
82         Paths {
83             FullMesh {
84                 profile normal
85                 ABOVE.IN
86                 ABOVE.COND
87                 ABOVE.DRILL
88             }
89             ABOVE.IN <-> AT.IN profile normal
90             ABOVE.COND <-> AT.COND profile normal
91             ABOVE.DRILL <-> AT.DRILL profile normal

```

---

---

```

92     ABOVE.DRILL <-> OUT.DRILL profile normal
93     OUT.DRILL <-> AT.DRILL profile normal
94   }
95 }
96 }
97
98 Resource UnloadRobot {
99   CL: Clamp
100  XY: XYMotor {
101    AxisPositions {
102      X (COND, DRILL, OUT)
103      Y (ABOVE, AT)
104    }
105    SymbolicPositions {
106      ABOVE.OUT (X.OUT, Y.ABOVE)
107      ABOVE.COND (X.COND, Y.ABOVE)
108      ABOVE.DRILL (X.DRILL, Y.ABOVE)
109      AT.OUT (X.OUT, Y.AT)
110      AT.COND (X.COND, Y.AT)
111      AT.DRILL (X.DRILL, Y.AT)
112      OUT.DRILL (X.DRILL)
113    }
114    Profiles (normal)
115    Paths {
116      FullMesh {
117        profile normal
118        ABOVE.COND
119        ABOVE.DRILL
120        ABOVE.OUT
121      }
122      ABOVE.COND <-> AT.COND profile normal
123      ABOVE.DRILL <-> AT.DRILL profile normal
124      ABOVE.DRILL <-> OUT.DRILL profile normal
125      OUT.DRILL <-> AT.DRILL profile normal
126      ABOVE.OUT <-> AT.OUT profile normal
127    }
128  }
129 }

```



---

## B LSAT code: Settings specification for Twilight system

```
1 import "twilight.machine"
2
3 LoadRobot.CL {
4   Timings {
5     clamp = Pert(min=0.1, max=1, mode=0.250, gamma=10)
6     unclamp = 0.200
7   }
8 }
9
10 LoadRobot.XY {
11   Axis X {
12     Profiles {
13       normal (V = 1, A = 8, J = 20)
14     }
15     Positions {
16       IN = 1
17       COND = 2
18       DRILL = 3
19     }
20   }
21   Axis Y {
22     Profiles {
23       normal (V = 2, A = 15, J = 35)
24     }
25     Positions {
26       ABOVE = 0
27       OUT.DRILL = 0.8
28       AT = 2
29     }
30   }
31 }
32
33 UnloadRobot.CL {
34   Timings {
35     clamp = 0.25
36
37     unclamp = 0.200
38   }
39 }
40
41 UnloadRobot.XY {
42   Axis X {
43     Profiles {
44       normal (V = 8, A = 8, J = 20)
45     }
46     Positions {
```

---

```

47     COND = 2
48     DRILL = 3
49     OUT = 4
50   }
51 }
52 Axis Y {
53   Profiles {
54     normal (V = 15, A = 15, J = 35)
55   }
56   Positions {
57     ABOVE = 0
58     OUT_DRILL = 0.8
59     AT = 2
60   }
61 }
62 }
63
64 Conditioner.CL {
65   Timings {
66     clamp = 0.250
67     unclamp = 0.200
68   }
69 }
70
71 Conditioner.CD {
72   Timings {
73     condition = 5.0
74   }
75 }
76
77 Drill.CL {
78   Timings {
79     clamp = 0.250
80     unclamp = 0.200
81   }
82 }
83
84 Drill.DL{
85   Timings {
86     on = 0.5
87     off = 0.5
88   }
89 }
90
91 Drill.ZR {
92   Axis Z {
93     Profiles {
94     normal (V = 0.1, A = 1, J = 5)
95     slow (V = 0.1, A = 0.1, J = 1.0)

```

---

---

```
96     }  
97     Positions {  
98         UP = 100  
99         DOWN = 0  
100    }  
101 }  
102 }
```

---

## C LSAT code: Activity specification for Twilight system

```
1 import "twilight.machine"
2
3
4 activity LR_PickPrdFromInput {
5     prerequisites {
6         LoadRobot.XY at ABOVE.IN
7     }
8     actions {
9         C1: claim LoadRobot
10        R1: release LoadRobot
11        A1: move LoadRobot.XY to AT.IN with speed profile normal
12        A2: LoadRobot.CL.clamp
13        A3: move LoadRobot.XY to ABOVE.IN with speed profile normal
14    }
15    action flow {
16        C1 -> A1 -> A2 -> A3 -> R1
17    }
18 }
19
20 activity LR_PutPrdOnCond {
21     prerequisites {
22         LoadRobot.XY at ABOVE.IN
23     }
24     actions {
25         C1: claim LoadRobot
26         C2: claim Conditioner
27         R1: release LoadRobot
28         R2: release Conditioner
29         C3: claim CollisionArea
30         R3: release CollisionArea
31         A1: move LoadRobot.XY to ABOVE.COND with speed profile normal
32         A2: move LoadRobot.XY to AT.COND with speed profile normal
33         A3: Conditioner.CL.clamp
34         A4: LoadRobot.CL.unclamp
35         A5: move LoadRobot.XY to ABOVE.COND with speed profile normal
36         A6: move LoadRobot.XY to ABOVE.IN with speed profile normal
37     }
38     action flow {
39         C1 -> C3 -> A1 -> A2 -> C2 -> A3 -> A4 -> R2 -> A5 -> A6 -> R3
40         ->R1
41     }
42 }
43 activity LR_PutPrdOnDrill {
44     prerequisites {
45         LoadRobot.XY at ABOVE.IN
```

---

```

46 }
47 actions {
48     C1: claim LoadRobot
49     C2: claim Drill
50     R1: release LoadRobot
51     R2: release Drill
52     C3: claim CollisionArea
53     R3: release CollisionArea
54     A2: move LoadRobot.XY to ABOVE_DRILL with speed profile normal
55     A3: move LoadRobot.XY to OUT_DRILL with speed profile normal
56     A4: move LoadRobot.XY to AT_DRILL with speed profile normal
57     A5: Drill.CL.clamp
58     A6: LoadRobot.CL.unclamp
59     A7: move LoadRobot.XY to OUT_DRILL with speed profile normal
60     A8: move LoadRobot.XY to ABOVE_DRILL with speed profile normal
61     A10: move LoadRobot.XY to ABOVE_IN with speed profile normal
62 }
63 action flow {
64     C1 -> C3 -> A2 -> A3 -> A4 -> C2 -> A5 -> A6 -> A7 -> R2 -> A8
        -> A10 -> R3-> R1
65 }
66 }
67
68 activity LR.PickPrdFromCond {
69     prerequisites {
70         LoadRobot.XY at ABOVE_IN
71     }
72     actions {
73         C1: claim LoadRobot
74         C2: claim Conditioner
75         R1: release LoadRobot
76         R2: release Conditioner
77         C3: claim CollisionArea
78         R3: release CollisionArea
79         A1: move LoadRobot.XY to ABOVE_COND with speed profile normal
80         A2: move LoadRobot.XY to AT_COND with speed profile normal
81         A3: Conditioner.CL.unclamp
82         A4: LoadRobot.CL.clamp
83         A5: move LoadRobot.XY to ABOVE_COND with speed profile normal
84         A6: move LoadRobot.XY to ABOVE_IN with speed profile normal
85     }
86     action flow {
87         C1 -> C3 -> A1 -> A2 -> C2 -> A4 -> A3 -> R2 -> A5 -> A6 -> R3->
            R1
88     }
89 }
90
91 activity LR.PickPrdFromDrill {
92     prerequisites {

```

---

---

```

93     LoadRobot.XY at ABOVE.IN
94 }
95 actions {
96     C1: claim LoadRobot
97     C2: claim Drill
98     R1: release LoadRobot
99     R2: release Drill
100    C3: claim CollisionArea
101    R3: release CollisionArea
102    A1: move LoadRobot.XY to ABOVE.COND with speed profile normal
103    A2: move LoadRobot.XY to ABOVE.DRILL with speed profile normal
104    A3: move LoadRobot.XY to OUT.DRILL with speed profile normal
105    A4: move LoadRobot.XY to AT.DRILL with speed profile normal
106    A5: Drill.CL.unclamp
107    A6: LoadRobot.CL.clamp
108    A7: move LoadRobot.XY to OUT.DRILL with speed profile normal
109    A8: move LoadRobot.XY to ABOVE.DRILL with speed profile normal
110    A9: move LoadRobot.XY to ABOVE.COND with speed profile normal
111    A10: move LoadRobot.XY to ABOVE.IN with speed profile normal
112 }
113 action flow {
114     C1 -> C3 -> A1 -> A2 -> A3 -> A4 -> C2 -> A6 -> A5 -> R2 -> A7
115     -> A8 -> A9 -> A10 -> R3-> R1
116 }

```

---

## D CIF code: Requirements

```
1 requirement req1:
2     location 10:
3         initial;
4         marked;
5         edge LR_PickPrdFromInput_1.A3 goto 11;
6     location 11:
7         marked;
8         edge LR_PutPrdOnCond_1.A3 goto 12;
9     location 12:
10        marked;
11 end
12
13 requirement req2:
14     location 10:
15         initial;
16         marked;
17         edge LR_PutPrdOnDrill_1.A5 goto 11;
18     location 11:
19         marked;
20         edge LR_PutPrdOnCond_2.A1 goto 12;
21     location 12:
22         marked;
23 end
```

---

## E Python code: Method I

```
1 #Method1: Crude Method where all possible sequences are individually
  checked
2
3 import networkx as nx
4 from itertools import product
5 import subprocess
6 import copy
7 import time
8 from pathlib import Path
9 from ConvertToAutomata import ConvertToAutomata
10 from subseqchecker import writetofile
11
12
13 fileDirReq=Path("se-software-cmdline-win-win-x64-r9682/bin/
    re_final_report.cif")
14 req=open(fileDirReq,"r")
15 Z_main=req.read()
16 req.close()
17
18 #Start: Separating important and non-important activities
19
20 Z_trav=Z_main.splitlines()
21 imp_act=[]
22 for line in Z_trav:
23     if "edge " in line:
24         wrd_arr=line.split()
25         imp_act.append(wrd_arr[wrd_arr.index("edge")+1].split(".")[0])
26
27 #Start: Extraction of LSAT variables
28
29 words_list_master=[]
30 comment_var=False
31 with open('twilight.activity','r') as file:
32
33     # reading each line
34     for line in file:
35
36         # reading each word
37         for word in line.split():
38
39             #Remove comments
40             if word.startswith("//"):
41                 break
42
43             if word.startswith("/"):
44                 comment_var=True
```



---

```

45
46     if "*" in word:
47         comment_var=False
48         to_remove=word.split("/")
49         new_word=to_remove[1]
50         if new_word:
51             words_list_master.append(new_word)
52         continue
53
54     # storing the words
55     if comment_var==False:
56         if ":" in word:
57             to_remove=word.split(":")
58             new_word=to_remove[0]
59             if new_word:
60                 words_list_master.append(new_word)
61             words_list_master.append(":")
62             new_word=to_remove[1]
63             if new_word:
64                 words_list_master.append(new_word)
65             continue
66
67         if "->" in word:
68             to_remove=word.split("->")
69             new_word=to_remove[0]
70             if new_word:
71                 words_list_master.append(new_word)
72             words_list_master.append("->")
73             new_word=to_remove[1]
74             if new_word:
75                 words_list_master.append(new_word)
76             continue
77
78
79         words_list_master.append(word)
80
81 main_stack=[]
82 activities_list=[]
83 for word in words_list_master:
84     main_stack.append(word)
85
86     if word=="}":
87         data_stack=[]
88         main_stack.pop()
89         while(True):
90             x=main_stack.pop()
91             if x=="{":
92                 break
93             data_stack.append(x)

```

---

---

```

94
95     # print(data_Stack)
96
97     if main_Stack[-1]=="actions":
98         if data_Stack:
99             temp_graph=nx.DiGraph()
100             rev_data_Stack=data_Stack[::-1]
101
102             for word2 in range(len(rev_data_Stack)):
103                 if rev_data_Stack[word2]==":":
104                     node_name=rev_data_Stack[word2-1]
105
106                     if rev_data_Stack[word2+1]=="claim":
107                         type_var="claim"
108                         resource_var=rev_data_Stack[word2+2]
109                     elif rev_data_Stack[word2+1]=="release":
110                         type_var="release"
111                         resource_var=rev_data_Stack[word2+2]
112                     elif rev_data_Stack[word2+1]=="move":
113                         type_var="action"
114                         resource_var=rev_data_Stack[word2+2].
115                             split(".")[0]
116                     else:
117                         type_var="action"
118                         resource_var=rev_data_Stack[word2+1].
119                             split(".")[0]
120
121                 temp_graph.add_node(node_name, resource=
122                     resource_var, type=type_var)
123                 empty_act=False
124             else:
125                 empty_act=True
126             # print(temp_graph.nodes)
127
128     if main_Stack[-1]=="flow":
129         if data_Stack:
130             rev_data_Stack=data_Stack[::-1]
131
132             for word2 in range(len(rev_data_Stack)-1):
133                 if rev_data_Stack[word2]=="->":
134                     if rev_data_Stack[word2+1].startswith("|"):
135                         sync_nodes=[]
136                         for word3 in range(len(rev_data_Stack)):
137                             if rev_data_Stack[word3]==
138                                 rev_data_Stack[word2+1]:
139                                 if word3<len(rev_data_Stack)-1:
140                                     if rev_data_Stack[word3+1]=="
141                                         ->":
142                                         sync_nodes.append(

```

---

---

```

rev_data_Stack[word3
+2])
138     for sn in sync_nodes:
139         temp_graph.add_edge(rev_data_Stack[
word2-1],sn)
140
141
142     else:
143         if not rev_data_Stack[word2-1].startswith
(" "):
144             temp_graph.add_edge(rev_data_Stack[
word2-1],rev_data_Stack[word2+1])
145
146         empty_act=False
147     else:
148         empty_act=True
149
150     if main_Stack[-2]=="activity" and not empty_act:
151         activities_list.append([copy.deepcopy(main_Stack[-1]),
copy.deepcopy(temp_graph)])
152
153
154 activities_list_names=[element[0] for element in activities_list]
155 #End: Extraction of LSAT variables
156
157 synths_array=[]
158 time_var=[]
159
160 size_of_seq=1 #Max size of sequence
161
162 #Start: Initial List of sequences to visited
163 arr_temp=[]
164
165 new_arr_names_only=[]
166 for i in range(size_of_seq):
167     arr_temp=list(p for p in product(activities_list_names, repeat=i
+1))
168     new_arr_names_only=new_arr_names_only+arr_temp
169
170 arr_names=[]
171
172 for i in new_arr_names_only:
173     instance_list=[]
174     list_to_be_passed=[]
175     list_to_be_passed_names=[]
176     for j in i:
177         instance_list.append(j)
178         list_to_be_passed_names.append(j+"_"+str(instance_list.count(
j)))

```

---

---

```

179     arr_names.append(list_to_be_passed_names)
180
181
182 # End: Initial List of sequences to visited
183
184
185 unimp_act=[]
186 for i in activities_list:
187     for j in range(size_of_seq):
188         if (i[0]+"_"+str(j+1)) not in imp_act:
189             unimp_act.append(i[0]+"_"+str(j+1))
190
191
192 #End: Separating important and non-important activities
193 finalReq="requirement req:\n"
194 cntr=0
195 seq_accepted_names=[]
196 no_of_synths=0
197 synthPath=Path( 'se-software-commandline-win-win-x64-r9682/bin/
198                 cif3datasynth.bat' )
199 start=time.time()
200 for i in arr_names:
201     list_to_be_passed_names=i
202     list_to_be_passed=[]
203
204     for j in range(len(i)):
205         list_to_be_passed.append([i[j], activities_list[
206             activities_list_names.index(new_arr_names_only[arr_names.
207                 index(i)][j) ][1]])
208
209
210 Y=ConvertToAutomata(list_to_be_passed)
211 X=Y.stringtowrite()
212 fileDirPlant=Path("se-software-commandline-win-win-x64-r9682/bin/
213                   temp_.cif")
214 Z=""
215 Z_trav=Z_main.splitlines()
216 for line in Z_trav:
217     if "edge " in line:
218         wrd_arr=line.split()
219         act_to_remove=wrd_arr[wrd_arr.index("edge")+1].split(".")
220         [0]
221
222         act_available=[item[0] for item in list_to_be_passed]
223         if act_to_remove in act_available:
224             Z=Z+line+"\n"
225         else:
226             if "edge " not in Z_trav[Z_trav.index(line)+1] and "
227                 initial;" not in Z_trav[Z_trav.index(line)+1] and
228                 "marked;" not in Z_trav[Z_trav.index(line)+1]:

```

---

---

```

221         if "location " in Z_trav[Z_trav.index(line)-1]:
222             pos=Z.rfind(':')
223             Z=Z[:pos]+";"
224         else:
225             Z=Z+line+"\n"
226
227
228     X=X+"\n"+Z
229     f=open(fileDirPlant,"w")
230     f.write(X)
231     f.close()
232     # Supervisory synthesis process
233     synth = subprocess.run(
234         [synthPath.absolute().as_posix(),
235          fileDirPlant.absolute().as_posix()],
236         capture_output=True,
237         text=True
238     )
239     no_of_synths+=1
240     print("Sequence visited: "+str(no_of_synths))
241     if "ERROR" not in synth.stderr:
242         seq_accepted_names.append(list_to_be_passed_names)
243
244     end=time.time()
245
246     print("Total synths:"+str(no_of_synths))
247     print("Total time:"+str(end-start))
248
249     finalReq=finalReq+"\tlocation L0:\n\t\tinitial;\n"
250     cntr=1
251
252     for i in [item[0] for item in seq_accepted_names]:
253         finalReq=finalReq+"\t\tedge "+i+" goto LSeq"+str(cntr)+"_1;\n"
254         cntr+=1
255
256     cntr=1
257     for i in seq_accepted_names:
258         for j in i:
259             if i.index(j)>0:
260                 finalReq=finalReq+"\tlocation LSeq"+str(cntr)+"_"
261                 +str(i.index(j))+":\n"
262                 finalReq=finalReq+"\t\tedge "+j+" goto LSeq"+str(
263                     cntr)+"_"+str(i.index(j)+1)+";\n"
264
265                 finalReq=finalReq+"\tlocation LSeq"+str(cntr)+"_"
266                 +str(i.index(j)+1)+":\n\t\tmarked;\n\n"
267                 cntr+=1

```

---

---

```
266 finalReq=finalReq+"\nend"
267
268 fileDirfinal=Path("se-software-cmdline-win-win-x64-r9682/bin/
    req-final_method_1.cif")
269 fin=open(fileDirfinal,"w")
270 fin.write(finalReq)
271 fin.close()
```

---

## F Python code: Method II

```
1 #Method 2: Improved method where sequences with addition of
2 #unimportant activities are not checked
3
4 import networkx as nx
5 import subprocess
6 import copy
7 import time
8 from pathlib import Path
9 from itertools import product
10 from subseqchecker import subseqchecker
11 from subseqchecker import diff
12 from subseqchecker import subseqcheckerend
13 from ConvertToAutomata import ConvertToAutomata
14
15
16 fileDirReq=Path("se-software-cmdline-win-win-x64-r9682/bin/
17 re_final_report.cif")
18 req=open(fileDirReq,"r")
19 Z_main=req.read()
20 req.close()
21
22 #Start: Separating important and non-important activities
23
24 Z_trav=Z_main.splitlines()
25 imp_act=[]
26 for line in Z_trav:
27     if "edge " in line:
28         wrd_arr=line.split()
29         imp_act.append(wrd_arr[wrd_arr.index("edge")+1].split(".")[0])
30
31 #Start: Extraction of LSAT variables
32
33 words_list_master=[]
34 comment_var=False
35 with open('twilight.activity','r') as file:
36
37     # reading each line
38     for line in file:
39
40         # reading each word
41         for word in line.split():
42
43             #Remove comments
44             if word.startswith("//"):
45                 break
```

---

```

46
47     if word.startswith("/"):
48         comment_var=True
49
50     if "*/" in word:
51         comment_var=False
52         to_remove=word.split("*/")
53         new_word=to_remove[1]
54         if new_word:
55             words_list_master.append(new_word)
56         continue
57
58     # storing the words
59     if comment_var==False:
60         if ":" in word:
61             to_remove=word.split(":")
62             new_word=to_remove[0]
63             if new_word:
64                 words_list_master.append(new_word)
65             words_list_master.append(":")
66             new_word=to_remove[1]
67             if new_word:
68                 words_list_master.append(new_word)
69             continue
70
71         if "->" in word:
72             to_remove=word.split("->")
73             new_word=to_remove[0]
74             if new_word:
75                 words_list_master.append(new_word)
76             words_list_master.append("->")
77             new_word=to_remove[1]
78             if new_word:
79                 words_list_master.append(new_word)
80             continue
81
82
83         words_list_master.append(word)
84
85 main_stack=[]
86 activities_list=[]
87 for word in words_list_master:
88     main_stack.append(word)
89
90     if word=="}":
91         data_stack=[]
92         main_stack.pop()
93         while(True):
94             x=main_stack.pop()

```

---



---

```

95         if x=="{":
96             break
97         data_stack.append(x)
98
99     # print(data_stack)
100
101     if main_stack[-1]=="actions":
102         if data_stack:
103             temp_graph=nx.DiGraph()
104             rev_data_stack=data_stack[::-1]
105
106             for word2 in range(len(rev_data_stack)):
107                 if rev_data_stack[word2]==":":
108                     node_name=rev_data_stack[word2-1]
109
110                     if rev_data_stack[word2+1]=="claim":
111                         type_var="claim"
112                         resource_var=rev_data_stack[word2+2]
113                     elif rev_data_stack[word2+1]=="release":
114                         type_var="release"
115                         resource_var=rev_data_stack[word2+2]
116                     elif rev_data_stack[word2+1]=="move":
117                         type_var="action"
118                         resource_var=rev_data_stack[word2+2].
119                             split(".")[0]
120                     else:
121                         type_var="action"
122                         resource_var=rev_data_stack[word2+1].
123                             split(".")[0]
124
125                 temp_graph.add_node(node_name, resource=
126                     resource_var, type=type_var)
127                 empty_act=False
128             else:
129                 empty_act=True
130             # print(temp_graph.nodes)
131
132     if main_stack[-1]=="flow":
133         if data_stack:
134             rev_data_stack=data_stack[::-1]
135
136             for word2 in range(len(rev_data_stack)-1):
137                 if rev_data_stack[word2]=="->":
138                     if rev_data_stack[word2+1].startswith("|"):
139                         sync_nodes=[]
140                         for word3 in range(len(rev_data_stack)):
141                             if rev_data_stack[word3]==
142                                 rev_data_stack[word2+1]:
143                                 if word3<len(rev_data_stack)-1:

```

---

---

```

140         if rev_data_Stack[word3+1]=="
141             ->":
142             sync_nodes.append(
143                 rev_data_Stack[word3
144                     +2])
145         for sn in sync_nodes:
146             temp_graph.add_edge(rev_data_Stack[
147                 word2-1],sn)
148     else:
149         if not rev_data_Stack[word2-1].startswith
150             ("|"):
151             temp_graph.add_edge(rev_data_Stack[
152                 word2-1],rev_data_Stack[word2+1])
153     empty_act=False
154     else:
155         empty_act=True
156     if main_Stack[-2]=="activity" and not empty_act:
157         activities_list.append([copy.deepcopy(main_Stack[-1]),
158             copy.deepcopy(temp_graph)])
159 activities_list_names=[element[0] for element in activities_list]
160 #End: Extraction of LSAT variables
161 size_of_seq=1 #max size of sequence
162 #Start: Initial List of sequences to visited
163 arr_temp=[]
164 new_arr_names_only=[]
165 for i in range(size_of_seq):
166     arr_temp=list(p for p in product(activities_list_names, repeat=i
167         +1))
168     new_arr_names_only=new_arr_names_only+arr_temp
169 arr_names=[]
170 for i in new_arr_names_only:
171     instance_list=[]
172     list_to_be_passed=[]
173     list_to_be_passed_names=[]
174     for j in i:
175         instance_list.append(j)

```

---

---

```

181         list_to_be_passed_names.append(j+"_"+str(instance_list.count(
182             j)))
183     arr_names.append(list_to_be_passed_names)
184
185     # End: Initial List of sequences to visited
186
187     unimp_act=[]
188     for i in activities_list:
189         for j in range(size_of_seq):
190             if (i[0]+"_"+str(j+1)) not in imp_act:
191                 unimp_act.append(i[0]+"_"+str(j+1))
192
193
194     #End: Separating important and non-important activities
195     indexes_to_be_visited=list(range(len(arr_names)))
196     len_cntr=[]
197     for i in range(1, size_of_seq+1):
198         len_cntr.append(pow(len(activities_list_names), i))
199
200
201     finalReq="requirement req:\n"
202     cntr=0
203     seq_accepted_names=[]
204     no_of_synths=0
205     synthPath=Path('se-software-cmdline-win-win-x64-r9682/bin/
206         cif3datasynth.bat')
207     start=time.time()
208     for i in arr_names:
209         list_to_be_passed_names=i
210
211         if arr_names.index(i) in indexes_to_be_visited:
212             indexes_to_be_visited.remove(arr_names.index(i))
213
214             for r in len_cntr:
215                 if arr_names.index(i)<r:
216                     reg=r
217                     break
218
219             list_to_be_passed=[]
220
221             for j in range(len(i)):
222                 list_to_be_passed.append([i[j], activities_list[
223                     activities_list_names.index(new_arr_names_only[
224                         arr_names.index(i)][j] )][1]])
225
226             Y=ConvertToAutomata(list_to_be_passed)
227             X=Y.stringtowrite()
228             fileDirPlant=Path("se-software-cmdline-win-win-x64-r9682/bin/

```

---

---

```

temp_.cif")
226 Z=""
227
228 Z_trav=Z_main.splitlines()
229 for line in Z_trav:
230     if "edge " in line:
231         wrd_arr=line.split()
232         act_to_remove=wrд_arr[wrд_arr.index("edge")+1].split(
                ".")[0]
233
234         act_available=[item[0] for item in list_to_be_passed]
235         if act_to_remove in act_available:
236             Z=Z+line+"\n"
237         else:
238             if "edge " not in Z_trav[Z_trav.index(line)+1]
                and "initial;" not in Z_trav[Z_trav.index(line)
                +1] and "marked;" not in Z_trav[Z_trav.index(
                line)+1]:
239                 if "location " in Z_trav[Z_trav.index(line)
                -1]:
240                     pos=Z.rfind(':')
241                     Z=Z[:pos]+";"
242             else:
243                 Z=Z+line+"\n"
244
245 X=X+"\n"+Z
246 f=open(fileDirPlant,"w")
247 f.write(X)
248 f.close()
249
250 # Supervisory synthesis process
251 synth = subprocess.run(
252     [synthPath.absolute().as_posix(),
        fileDirPlant.absolute().as_posix()],
253     capture_output=True,
254     shell=True,
255     text=True
256 )
257 no_of_synths+=1
258 print("Sequence visited:"+str(no_of_synths))
259 if "ERROR" not in synth.stderr:
260     seq_accepted_names.append(list_to_be_passed_names)
261     indexes_temp=list(indexes_to_be_visited)
262     if len(indexes_to_be_visited)==0:
263         break
264     for k in indexes_temp:
265         if k>=reg:
266             if subseqchecker(arr_names[k],
                list_to_be_passed_names):

```

---

---

```

267         if set(diff(arr_names[k],
268                    list_to_be_passed_names)).issubset(set(
269                    unimp_act)):
270             seq_accepted_names.append(arr_names[k])
271             indexes_to_be_visited.remove(k)
272
273         # print(seq_accepted_names)
274
275     else:
276         indexes_temp=list(indexes_to_be_visited)
277         if len(indexes_to_be_visited)==0:
278             break
279         for k in indexes_temp:
280             if k>=reg:
281                 if subseqcheckerend(list_to_be_passed_names,
282                                    arr_names[k]):
283                     if set(diff(arr_names[k],
284                                list_to_be_passed_names)).issubset(set(
285                                unimp_act)):
286                         indexes_to_be_visited.remove(k)
287
288 end=time.time()
289
290 print("Total synths:"+str(no_of_synths))
291 print("Total time:"+str(end-start))
292
293 finalReq=finalReq+"\tlocation L0:\n\t\tinitial;\n"
294 cntr=1
295
296 for i in [item[0] for item in seq_accepted_names]:
297     finalReq=finalReq+"\t\tedge "+i+" goto LSeq"+str(cntr)+"_1;\n"
298     cntr+=1
299
300 cntr=1
301 for i in seq_accepted_names:
302     for j in i:
303         if i.index(j)>0:
304             finalReq=finalReq+"\tlocation LSeq"+str(cntr)+"_"
305             +str(i.index(j))+":\n"
306             finalReq=finalReq+"\t\tedge "+j+" goto LSeq"+str(
307             cntr)+"_" +str(i.index(j)+1)+":\n"
308
309 finalReq=finalReq+"\tlocation LSeq"+str(cntr)+"_" +str(i.
310 index(j)+1)+":\n\t\tmarked;\n\n"
311 cntr+=1

```

---

---

```
308
309 finalReq=finalReq+"\nend"
310
311 fileDirfinal=Path("se-software-cmdline-win-win-x64-r9682/bin/
    req-final.method_2.cif")
312 fin=open(fileDirfinal,"w")
313 fin.write(finalReq)
314 fin.close()
```

---

## G Python code: Method III

```
1 #Method 3: Improved method where actions which are not claim release
2 #or connecting activities are ignored
3
4 import networkx as nx
5 import subprocess
6 import copy
7 import time
8 from pathlib import Path
9 from itertools import product
10 from ConvertToAutomata import ConvertToAutomata
11 from subseqchecker import subseqchecker
12 from subseqchecker import subseqcheckerend
13 from subseqchecker import diff
14
15 fileDirReq=Path("se-software-cmdline-win-win-x64-r9682/bin/
16     re_final_report.cif")
17 req=open(fileDirReq,"r")
18 Z_main=req.read()
19 req.close()
20
21 #Start: Separating important and non-important activities
22 Z_trav=Z_main.splitlines()
23 imp_act=[]
24 imp_action={}
25
26 for line in Z_trav:
27     if "edge " in line:
28         wrd_arr=line.split()
29         imp_act.append(wrd_arr[wrd_arr.index("edge")+1].split(".")[0])
30         if wrd_arr[wrd_arr.index("edge")+1].split(".")[0] in
31             imp_action:
32             imp_action[wrd_arr[wrd_arr.index("edge")+1].split(".")[0]].append( wrd_arr[wrd_arr.index("edge")+1].split(".")[1])
33         else:
34             imp_action[wrd_arr[wrd_arr.index("edge")+1].split(".")[0]]=[]
35             imp_action[wrd_arr[wrd_arr.index("edge")+1].split(".")[0]].append( wrd_arr[wrd_arr.index("edge")+1].split(".")[1])
36
37 #Start: Extraction of LSAT variables
38 words_list_master=[]
39 comment_var=False
40 with open('twilight.activity','r') as file:
```

---

```
40
41 # reading each line
42 for line in file:
43
44     # reading each word
45     for word in line.split():
46
47         #Remove comments
48         if word.startswith("//"):
49             break
50
51         if word.startswith("/*"):
52             comment_var=True
53
54         if "*/" in word:
55             comment_var=False
56             to_remove=word.split("*/")
57             new_word=to_remove[1]
58             if new_word:
59                 words_list_master.append(new_word)
60             continue
61
62         # storing the words
63         if comment_var==False:
64             if ":" in word:
65                 to_remove=word.split(":")
66                 new_word=to_remove[0]
67                 if new_word:
68                     words_list_master.append(new_word)
69                 words_list_master.append(":")
70                 new_word=to_remove[1]
71                 if new_word:
72                     words_list_master.append(new_word)
73                 continue
74
75             if "->" in word:
76                 to_remove=word.split("->")
77                 new_word=to_remove[0]
78                 if new_word:
79                     words_list_master.append(new_word)
80                 words_list_master.append("->")
81                 new_word=to_remove[1]
82                 if new_word:
83                     words_list_master.append(new_word)
84                 continue
85
86
87         words_list_master.append(word)
88
```

---



---

```

89 main_Stack=[]
90 activities_list=[]
91 for word in words_list_master:
92     main_Stack.append(word)
93
94     if word=="}":
95         data_Stack=[]
96         main_Stack.pop()
97         while(True):
98             x=main_Stack.pop()
99             if x=="{":
100                 break
101             data_Stack.append(x)
102
103         # print(data_Stack)
104
105         if main_Stack[-1]=="actions":
106             if data_Stack:
107                 temp_graph=nx.DiGraph()
108                 rev_data_Stack=data_Stack[::-1]
109
110                 for word2 in range(len(rev_data_Stack)):
111                     if rev_data_Stack[word2]==":":
112                         node_name=rev_data_Stack[word2-1]
113
114                         if rev_data_Stack[word2+1]=="claim":
115                             type_var="claim"
116                             resource_var=rev_data_Stack[word2+2]
117                         elif rev_data_Stack[word2+1]=="release":
118                             type_var="release"
119                             resource_var=rev_data_Stack[word2+2]
120                         elif rev_data_Stack[word2+1]=="move":
121                             type_var="action"
122                             resource_var=rev_data_Stack[word2+2].
123                                 split(".")[0]
124                         else:
125                             type_var="action"
126                             resource_var=rev_data_Stack[word2+1].
127                                 split(".")[0]
128
129                             temp_graph.add_node(node_name,name=node_name,
130                                     resource=resource_var,type=type_var)
131                             empty_act=False
132                         else:
133                             empty_act=True
134                         # print(temp_graph.nodes)
135
136         if main_Stack[-1]=="flow":
137             if data_Stack:

```

---

---

```

135         rev_data_stack=data_stack[:-1]
136
137         for word2 in range(len(rev_data_stack)-1):
138             if rev_data_stack[word2]=="->":
139                 if rev_data_stack[word2+1].startswith("|"):
140                     sync_nodes=[]
141                     for word3 in range(len(rev_data_stack)):
142                         if rev_data_stack[word3]==
143                             rev_data_stack[word2+1]:
144                             if word3<len(rev_data_stack)-1:
145                                 if rev_data_stack[word3+1]=="
146                                     ->":
147                                     sync_nodes.append(
148                                         rev_data_stack[word3
149                                             +2])
150
151                     for sn in sync_nodes:
152                         temp_graph.add_edge(rev_data_stack[
153                             word2-1],sn)
154
155             else:
156                 if not rev_data_stack[word2-1].startswith
157                     ("|"):
158                     temp_graph.add_edge(rev_data_stack[
159                         word2-1],rev_data_stack[word2+1])
160
161             empty_act=False
162         else:
163             empty_act=True
164
165         if main_stack[-2]=="activity" and not empty_act:
166             activities_list.append([copy.deepcopy(main_stack[-1]),
167                 copy.deepcopy(temp_graph)])
168
169     activities_list_names=[element[0] for element in activities_list]
170
171     #End: Extraction of LSAT variables
172
173     alternate_dag_imp=[]
174     alternate_dag_unimp=[]
175     for i in activities_list_names:
176         match_found=False
177         for j in imp_act:
178             temp_name=copy.deepcopy(j.rsplit('_',1)[0])
179             if temp_name==i:
180                 temp_graph1=copy.deepcopy(activities_list[
181                     activities_list_names.index(i)][1])
182                 nodes_list=list(temp_graph1.nodes)

```

---

---

```

175
176     for k in nodes_list:
177         if temp_graph1.nodes[k]['type']!= 'claim' and
178            temp_graph1.nodes[k]['type']!= 'release':
179             if k not in imp_action[j]:
180
181                 pred_node=list (temp_graph1.predecessors(k))
182                 succ_node=list (temp_graph1.successors(k))
183                 temp_graph1.remove_node(k)
184                 elist=[]
185                 for pred in pred_node:
186                     for succ in succ_node:
187                         elist.append((pred,succ))
188
189                 temp_graph1.add_edges_from( elist)
190
191                 alternate_dag_imp.append(copy.deepcopy(temp_graph1))
192                 match_found=True
193                 break
194
195 if not match_found:
196     alternate_dag_imp.append(-1)
197
198 for i in activities_list_names:
199     temp_graph1=copy.deepcopy( activities_list[activities_list_names.
200        index(i)][1])
201     nodes_list=list (temp_graph1.nodes)
202
203     for k in nodes_list:
204         if temp_graph1.nodes[k]['type']!= 'claim' and temp_graph1.
205            nodes[k]['type']!= 'release':
206
207             pred_node=list (temp_graph1.predecessors(k))
208             succ_node=list (temp_graph1.successors(k))
209             temp_graph1.remove_node(k)
210             elist=[]
211             for pred in pred_node:
212                 for succ in succ_node:
213                     elist.append((pred,succ))
214
215             temp_graph1.add_edges_from( elist)
216
217     alternate_dag_unimp.append(copy.deepcopy(temp_graph1))
218
219 size_of_seq=6 #max size of sequence
220 #Start: Initial List of sequences to visited

```

---

---

```

221 arr_temp=[]
222
223 new_arr_names_only=[]
224 for i in range(size_of_seq):
225     arr_temp=list(p for p in product(activities_list_names , repeat=i
226         +1))
227     new_arr_names_only=new_arr_names_only+arr_temp
228
229 arr_names=[]
230
231 for i in new_arr_names_only:
232     instance_list=[]
233     list_to_be_passed=[]
234     list_to_be_passed_names=[]
235     for j in i:
236         instance_list.append(j)
237         list_to_be_passed_names.append(j+"_"+str(instance_list.count(
238             j)))
239
240     arr_names.append(list_to_be_passed_names)
241
242 # End: Initial List of sequences to visited
243
244 #Start: Separating important and non-important activities
245
246 unimp_act=[]
247 for i in activities_list:
248     for j in range(size_of_seq):
249         if (i[0]+"_"+str(j+1)) not in imp_act:
250             unimp_act.append(i[0]+"_"+str(j+1))
251
252 #End: Separating important and non-important activities
253 indexes_to_be_visited=list(range(len(arr_names)))
254 len_cntr=[]
255 for i in range(1, size_of_seq+1):
256     len_cntr.append(pow(len(activities_list_names), i))
257
258 finalReq="requirement req:\n"
259 cntr=0
260 seq_accepted_names=[]
261
262 no_of_synths=0
263 synthPath=Path('se-software-cmdline-win-win-x64-r9682/bin/
264     cif3datasynth.bat')
265 start=time.time()
266 cntr=1
267 for i in arr_names:

```

---

---

```

267 list_to_be_passed_names=i
268
269 if arr_names.index(i) in indexes_to_be_visited:
270     indexes_to_be_visited.remove(arr_names.index(i))
271     for r in len_cntr:
272         if arr_names.index(i)<r:
273             reg=r
274             break
275
276 list_to_be_passed=[]
277
278 for j in range(len(i)):
279     if i[j] in imp_act:
280         temp_graph=copy.deepcopy(activities_list[
                activities_list_names.index(new_arr_names_only[
                arr_names.index(i)][j)])[1])
281         for k in list(temp_graph.nodes):
282             if temp_graph.nodes[k]['type']!= 'claim' and
                temp_graph.nodes[k]['type']!= 'release':
283                 if k not in imp_action[i[j]]:
284
285                     pred_node=list(temp_graph.predecessors(k)
                )
286                     succ_node=list(temp_graph.successors(k))
287                     temp_graph.remove_node(k)
288                     elist=[]
289                     for pred in pred_node:
290                         for succ in succ_node:
291                             elist.append((pred,succ))
292
293                     temp_graph.add_edges_from(elist)
294
295                 list_to_be_passed.append([i[j],copy.deepcopy(
                temp_graph)])
296         else:
297             list_to_be_passed.append([i[j],activities_list[
                activities_list_names.index(new_arr_names_only[
                arr_names.index(i)][j)])[1]])
298
299
300 Y=ConvertToAutomata(list_to_be_passed)
301 X=Y.stringtowrite()
302 fileDirPlant=Path("se-software-commandline-win-win-x64-r9682/bin/
                temp_new_3.cif")
303 Z=""
304
305 Z_trav=Z.main.splitlines()
306 for line in Z_trav:
307     if "edge " in line:

```

---

---

```

308 wrd_arr=line.split()
309 act_to_remove=wrd_arr[wrd_arr.index("edge")+1].split(
    ".")[0]
310
311 act_available=i
312 if act_to_remove in act_available:
313     Z=Z+line+"\n"
314 else:
315     if "edge " not in Z_trav[Z_trav.index(line)+1]
        and "initial;" not in Z_trav[Z_trav.index(line
        )+1] and "marked;" not in Z_trav[Z_trav.index(
        line)+1]:
316         if "location " in Z_trav[Z_trav.index(line)
            -1]:
317             pos=Z.rfind(':')
318             Z=Z[:pos]+";"
319     else:
320         Z=Z+line+"\n"
321
322
323 X=X+"\n"+Z
324 f=open(fileDirPlant,"w")
325 f.write(X)
326 f.close()
327 # Supervisory synthesis process
328 synth = subprocess.run(
329     [synthPath.absolute().as_posix(),
        fileDirPlant.absolute().as_posix()],
330     capture_output=True,
331     text=True
332 )
333 no_of_synths+=1
334 print("Sequence visited:"+str(no_of_synths))
335 if "ERROR" not in synth.stderr:
336     seq_accepted_names.append(list_to_be_passed_names)
337     indexes_temp=list(indexes_to_be_visited)
338     if len(indexes_to_be_visited)==0:
339         break
340     for k in indexes_temp:
341         if k>=reg:
342             if subseqchecker(arr_names[k],
                list_to_be_passed_names):
343                 if set(diff(arr_names[k],
                    list_to_be_passed_names)).issubset(set(
                    unimp_act)):
344                     seq_accepted_names.append(arr_names[k])
345                     indexes_to_be_visited.remove(k)
346
347     # print(seq_accepted_names)

```

---

---

```

348
349     else:
350         indexes_temp=list(indexes_to_be_visited)
351         if len(indexes_to_be_visited)==0:
352             break
353         for k in indexes_temp:
354             if k>=reg:
355                 if subseqcheckerend(list_to_be_passed_names ,
356                                     arr_names[k]):
357                     if set(diff(arr_names[k],
358                                 list_to_be_passed_names)).issubset(set(
359                                 unimp_act)):
360                         indexes_to_be_visited.remove(k)
361
362     end=time.time()
363
364     print("Total synths:"+str(no_of_synths))
365     print("Total time:"+str(end-start))
366
367
368     finalReq=finalReq+"\tlocation L0:\n\t\tinitial;\n"
369     cntr=1
370
371     for i in [item[0] for item in seq_accepted_names]:
372         finalReq=finalReq+"\t\tedge "+i+" goto LSeq"+str(cntr)+"_1;\n"
373         cntr+=1
374
375     cntr=1
376     for i in seq_accepted_names:
377         for j in i:
378             if i.index(j)>0:
379                 finalReq=finalReq+"\tlocation LSeq"+str(cntr)+"_"+str(i.
380                                     index(j))+":\n"
381                 finalReq=finalReq+"\t\tedge "+j+" goto LSeq"+str(cntr)+"_
382                                     "+str(i.index(j)+1)+";\n"
383
384                 finalReq=finalReq+"\tlocation LSeq"+str(cntr)+"_"+str(i.index(j)
385                                     +1)+":\n\t\tmarked;\n\n"
386                 cntr+=1
387
388     finalReq=finalReq+"\nend"
389
390     fileDirfinal=Path("se-software-cmdline-win-win-x64-r9682/bin/

```

---

---

```
    req_final_new.cif")
391 fin=open(fileDirfinal,"w")
392 fin.write(finalReq)
393 fin.close()
```



---

## H Python code: Method IV

```
1 import networkx as nx
2 import subprocess
3 import copy
4 import time
5 from multiprocessing import Pool
6 from pathlib import Path
7 from itertools import product
8 from ConvertToAutomata2 import ConvertToAutomata
9 from subseqchecker import subseqchecker
10 from subseqchecker import subseqcheckerend
11 from subseqchecker import diff
12 from subseqchecker import writetofile
13
14 if __name__ == '__main__':
15     fileDirReq=Path("se-software-cmdline-win-win-x64-r9682/bin/
16         re_final_report.cif")
17     req=open(fileDirReq,"r")
18     Z_main=req.read()
19     req.close()
20
21     #Start: Separating important and non-important activities
22     Z_trav=Z_main.splitlines()
23     imp_act=[]
24     imp_action={}
25
26     for line in Z_trav:
27         if "edge " in line:
28             wrd_arr=line.split()
29             imp_act.append(wrd_arr[wrd_arr.index("edge")+1].split(".")
30                 ) [0])
31             if wrd_arr[wrd_arr.index("edge")+1].split(".") [0] in
32                 imp_action:
33                 imp_action[wrd_arr[wrd_arr.index("edge")+1].split(".")
34                     ] [0]].append( wrd_arr[wrd_arr.index("edge")+1].
35                         split(".") [1])
36             else:
37                 imp_action[wrd_arr[wrd_arr.index("edge")+1].split(".")
38                     ] [0]]=[]
39                 imp_action[wrd_arr[wrd_arr.index("edge")+1].split(".")
40                     ] [0]].append( wrd_arr[wrd_arr.index("edge")+1].
41                         split(".") [1])
42
43     #Start: Extraction of LSAT variables
44     print(imp_action)
45     words_list_master=[]
46     comment_var=False
47     with open('twilight.activity','r') as file:
```

---

```

40
41 # reading each line
42 for line in file:
43
44     # reading each word
45     for word in line.split():
46
47         #Remove comments
48         if word.startswith("//"):
49             break
50
51         if word.startswith("/*"):
52             comment_var=True
53
54         if "*" in word:
55             comment_var=False
56             to_remove=word.split("/")
57             new_word=to_remove[1]
58             if new_word:
59                 words_list_master.append(new_word)
60             continue
61
62         # storing the words
63         if comment_var==False:
64             if ":" in word:
65                 to_remove=word.split(":")
66                 new_word=to_remove[0]
67                 if new_word:
68                     words_list_master.append(new_word)
69                 words_list_master.append(":")
70                 new_word=to_remove[1]
71                 if new_word:
72                     words_list_master.append(new_word)
73                 continue
74
75             if "->" in word:
76                 to_remove=word.split("->")
77                 new_word=to_remove[0]
78                 if new_word:
79                     words_list_master.append(new_word)
80                 words_list_master.append("->")
81                 new_word=to_remove[1]
82                 if new_word:
83                     words_list_master.append(new_word)
84                 continue
85
86         words_list_master.append(word)
87
88

```

---

---

```

89 main_stack=[]
90 activities_list=[]
91 for word in words_list_master:
92     main_stack.append(word)
93
94     if word=="}":
95         data_stack=[]
96         main_stack.pop()
97         while(True):
98             x=main_stack.pop()
99             if x=="{":
100                 break
101             data_stack.append(x)
102
103         # print(data_stack)
104
105         if main_stack[-1]=="actions":
106             if data_stack:
107                 temp_graph=nx.DiGraph()
108                 rev_data_stack=data_stack[::-1]
109
110                 for word2 in range(len(rev_data_stack)):
111                     if rev_data_stack[word2]==":":
112                         node_name=rev_data_stack[word2-1]
113
114                         if rev_data_stack[word2+1]=="claim":
115                             type_var="claim"
116                             resource_var=rev_data_stack[word2+2]
117                         elif rev_data_stack[word2+1]=="release":
118                             type_var="release"
119                             resource_var=rev_data_stack[word2+2]
120                         elif rev_data_stack[word2+1]=="move":
121                             type_var="action"
122                             resource_var=rev_data_stack[word2+2].
123                                 split(".")[0]
124                         else:
125                             type_var="action"
126                             resource_var=rev_data_stack[word2+1].
127                                 split(".")[0]
128
129                             temp_graph.add_node(node_name,name=
130                                 node_name,resource=resource_var,type=
131                                 type_var)
132                             empty_act=False
133                         else:
134                             empty_act=True
135                 # print(temp_graph.nodes)
136
137         if main_stack[-1]=="flow":

```

---

---

```

134         if data_Stack:
135             rev_data_Stack=data_Stack[::-1]
136
137             for word2 in range(len(rev_data_Stack)-1):
138                 if rev_data_Stack[word2]=="->":
139                     if rev_data_Stack[word2+1].startswith(" | "):
140                         sync_nodes=[]
141                         for word3 in range(len(rev_data_Stack
142 )):
143                             if rev_data_Stack[word3]==
144                                 rev_data_Stack[word2+1]:
145                                 if word3<len(rev_data_Stack)
146                                     -1:
147                                     if rev_data_Stack[word3
148                                         +1]=="->":
149                                         sync_nodes.append(
150                                             rev_data_Stack[
151                                                 word3+2])
152
153                             for sn in sync_nodes:
154                                 temp_graph.add_edge(
155                                     rev_data_Stack[word2-1],sn)
156
157                     else:
158                         if not rev_data_Stack[word2-1].
159                             startswith(" | "):
160                             temp_graph.add_edge(
161                                 rev_data_Stack[word2-1],
162                                 rev_data_Stack[word2+1])
163
164             empty_act=False
165         else:
166             empty_act=True
167
168         if main_Stack[-2]=="activity" and not empty_act:
169             activities_list.append([copy.deepcopy(main_Stack[-1])
170                                     ,copy.deepcopy(temp_graph)])
171
172     activities_list_names=[element[0] for element in activities_list]
173
174     #End: Extraction of LSAT variables
175
176     alternate_dag_imp=[]
177     alternate_dag_unimp=[]
178     for i in activities_list_names:
179         match_found=False
180         for j in imp_act:

```

---

---

```

171     temp_name=copy.deepcopy(j.rsplitt('_',1)[0])
172     if temp_name==i:
173         temp_graph1=copy.deepcopy(activities_list[
174             activities_list_names.index(i)][1])
175         nodes_list=list(temp_graph1.nodes)
176
177         for k in nodes_list:
178             if temp_graph1.nodes[k]['type']!='claim' and
179                temp_graph1.nodes[k]['type']!='release':
180
181                 if k not in imp_action[j]:
182
183                     pred_node=list(temp_graph1.predecessors(k
184                         ))
185                     succ_node=list(temp_graph1.successors(k))
186                     temp_graph1.remove_node(k)
187                     elist=[]
188                     for pred in pred_node:
189                         for succ in succ_node:
190                             elist.append((pred,succ))
191
192                     temp_graph1.add_edges_from(elist)
193
194                 alternate_dag_imp.append(copy.deepcopy(temp_graph1))
195                 match_found=True
196                 break
197
198     if not match_found:
199         alternate_dag_imp.append(-1)
200
201     for i in activities_list_names:
202         temp_graph1=copy.deepcopy(activities_list[
203             activities_list_names.index(i)][1])
204         nodes_list=list(temp_graph1.nodes)
205
206         for k in nodes_list:
207             if temp_graph1.nodes[k]['type']!='claim' and temp_graph1.
208                nodes[k]['type']!='release':
209
210                 pred_node=list(temp_graph1.predecessors(k))
211                 succ_node=list(temp_graph1.successors(k))
212                 temp_graph1.remove_node(k)
213                 elist=[]
214                 for pred in pred_node:
215                     for succ in succ_node:
216                         elist.append((pred,succ))
217
218                 temp_graph1.add_edges_from(elist)

```

---

---

```

215         alternate_dag_unimp.append(copy.deepcopy(temp_graph1))
216
217     size_of_seq=2 #max size of sequence
218
219     #Start: Initial List of sequences to visited
220
221     arr_temp=[]
222
223     new_arr_names_only=[]
224     for i in range(size_of_seq):
225         arr_temp=list(p for p in product(activities_list_names ,
226             repeat=i+1))
227         new_arr_names_only=new_arr_names_only+arr_temp
228
229     arr_names=[]
230
231     for i in new_arr_names_only:
232         instance_list=[]
233         list_to_be_passed=[]
234         list_to_be_passed_names=[]
235         for j in i:
236             instance_list.append(j)
237             list_to_be_passed_names.append(j+"_"+str(instance_list.
238                 count(j)))
239
240         arr_names.append(list_to_be_passed_names)
241
242     # End: Initial List of sequences to visited
243
244     #Start: Separating important and non-important activities
245
246     unimp_act=[]
247     for i in activities_list:
248         for j in range(size_of_seq):
249             if (i[0]+"_"+str(j+1)) not in imp_act:
250                 unimp_act.append(i[0]+"_"+str(j+1))
251
252     #End: Separating important and non-important activities
253     indexes_to_be_visited=list(range(len(arr_names)))
254     len_cntr=[]
255     for i in range(1,size_of_seq+1):
256         len_cntr.append(pow(len(activities_list_names),i))
257
258     finalReq="requirement req:\n"
259
260     seq_accepted_names=[]
261

```

---

---

```

262     no_of_synths=0
263     synthPath=Path( 'se-software-cmdline-win-win-x64-r9682/bin/
        cif3datasynth.bat' )
264
265     start=time.time()
266     cntr=1
267     for i in arr_names:
268         list_to_be_passed_names=i
269
270         if arr_names.index(i) in indexes_to_be_visited:
271
272             indexes_to_be_visited.remove(arr_names.index(i))
273             for r in len_cntr:
274                 if arr_names.index(i)<r:
275                     reg=r
276                     break
277
278             list_to_be_passed=[]
279
280             #         for j in range(len(i)):
281             #             if i[j] in imp_action:
282             #                 list_to_be_passed.append([i[j],
alternate_dag_imp[activities_list_names.index(new_arr_names_only[
arr_names.index(i)][j])]])
283             #             else:
284             #                 list_to_be_passed.append([i[j],
alternate_dag_unimp[activities_list_names.index(new_arr_names_only
[arr_names.index(i)][j])]])
285
286             for j in range(len(i)):
287                 list_to_be_passed.append([i[j], activities_list[
activities_list_names.index(new_arr_names_only[
arr_names.index(i)][j])][1]])
288
289             Y=ConvertToAutomata(list_to_be_passed)
290             X=Y.stringtowrite8(imp_action)
291             fileDirPlant=Path( "se-software-cmdline-win-win-x64-r9682/
        bin/temp_new_3.cif" )
292             Z=""
293
294             Z_trav=Z.main.splitlines()
295             for line in Z_trav:
296                 if "edge " in line:
297                     wrd_arr=line.split()
298                     act_to_remove=wrd_arr[wrd_arr.index("edge")+1].
split(".")[0]
299                     action_to_remove=wrd_arr[wrd_arr.index("edge")
+1].split(".")[1]
300                     act_available=i

```

---

---

```

301         if act_to_remove in act_available:
302             if action_to_remove in imp_action[
                 act_to_remove]:
303                 temp_line=line.replace('.', '_')
304                 Z=Z+temp_line+"\n"
305             else:
306                 if "edge " not in Z_trav[Z_trav.index(line)
                    +1] and "initial;" not in Z_trav[Z_trav.
                    index(line)+1] and "marked;" not in Z_trav
                    [Z_trav.index(line)+1]:
307                     if "location " in Z_trav[Z_trav.index(
                        line)-1]:
308                         pos=Z.rfind(':')
309                         Z=Z[:pos]+";"
310                 else:
311                     Z=Z+line+"\n"
312
313
314     X=X+"\n"+Z
315     f=open(fileDirPlant, "w")
316     f.write(X)
317     f.close()
318     # Supervisory synthesis process
319     synth = subprocess.run(
320         [synthPath.absolute().as_posix(),
          fileDirPlant.absolute().as_posix()
          ],
321         capture_output=True,
322         text=True
323     )
324     no_of_synths+=1
325     print("Sequence visited:"+str(no_of_synths))
326     if "ERROR" not in synth.stderr:
327         seq_accepted_names.append((list_to_be_passed_names,
          cntr, unimp_act))
328         cntr+=1
329         indexes_temp=list(indexes_to_be_visited)
330         if len(indexes_to_be_visited)==0:
331             break
332         for k in indexes_temp:
333             if k>=reg:
334                 if subseqchecker(arr_names[k],
          list_to_be_passed_names):
335                     if set(diff(arr_names[k],
          list_to_be_passed_names)).issubset(set
          (unimp_act)):
336                         indexes_to_be_visited.remove(k)
337                     if not subseqcheckerend(
          list_to_be_passed_names, arr_names

```

---



---

```

338         [k] :
339             seq_accepted_names.append((
340                 arr_names[k], cntr, unimp_act))
341             cntr+=1
342
343         # print(seq_accepted_names)
344
345     else:
346         indexes_temp=list(indexes_to_be_visited)
347         if len(indexes_to_be_visited)==0:
348             break
349         for k in indexes_temp:
350             if k>=reg:
351                 if subseqcheckerend(list_to_be_passed_names,
352                                     arr_names[k]):
353                     if set(diff(arr_names[k],
354                                 list_to_be_passed_names)).issubset(set
355                                 (unimp_act)):
356                         indexes_to_be_visited.remove(k)
357
358 end=time.time()
359
360 print("Total synths: "+str(no_of_synths))
361 print("Total time: "+str(end-start))
362
363 pool=Pool(processes=5)
364 temp_str_arr=list(pool.map(writetofile, seq_accepted_names))
365
366 finalReq=finalReq+"\tlocation L0:\n\t\tinitial;\n"
367
368 arr1=[element[0] for element in temp_str_arr]
369 arr2=[element[1] for element in temp_str_arr]
370 finalReq=finalReq+' '.join(arr1)
371 finalReq=finalReq+'\n'.join(arr2)
372 finalReq=finalReq+"\nend"
373
374 fileDirfinal=Path("se-software-cmdline-win-win-x64-r9682/bin/
375                   req_final_new.cif")
376 fin=open(fileDirfinal, "w")
377 fin.write(finalReq)
378 fin.close()

```

---

---

## I Python code: Method V

```
1
2
3 import networkx as nx
4 import itertools
5 import os.path
6 import subprocess
7 import copy
8 import time
9 from subseqchecker import subseqchecker
10 from subseqchecker import diff
11 from ConvertToAutomata import ConvertToAutomata
12 from activitySequenceExtractor import activitySequenceExtractor
13 #Start: Extraction of LSAT variables
14
15 fileDirReq=os.path.join("F:/Personal/TU Eindhoven/Post Registration/
    Grad Project/LSAT/se-software-commandline-win-win-x64-r9682/bin/", "req
    .cif")
16 req=open(fileDirReq, "r")
17 Z_main=req.read()
18 req.close()
19
20 Z_trav=Z_main.splitlines()
21 imp_act=[]
22 imp_action={}
23 for line in Z_trav:
24     if "edge " in line:
25         wrd_arr=line.split()
26         imp_act.append(wrd_arr[wrd_arr.index("edge")+1].split(".")[0])
27         if wrd_arr[wrd_arr.index("edge")+1].split(".")[0] in
            imp_action:
28             imp_action[wrd_arr[wrd_arr.index("edge")+1].split(".")[0]].append( wrd_arr[wrd_arr.index("edge")+1].split(".")[1])
29         else:
30             imp_action[wrd_arr[wrd_arr.index("edge")+1].split(".")[0]]=[]
31             imp_action[wrd_arr[wrd_arr.index("edge")+1].split(".")[0]].append( wrd_arr[wrd_arr.index("edge")+1].split(".")[1])
32
33
34
35 words_list_master=[]
36 comment_var=False
37 with open('Example.Convert.2.activity', 'r') as file:
38
```

---

```

39 # reading each line
40 for line in file:
41
42     # reading each word
43     for word in line.split():
44
45         #Remove comments
46         if word.startswith("//"):
47             break
48
49         if word.startswith("/*"):
50             comment_var=True
51
52         if "*/" in word:
53             comment_var=False
54             to_remove=word.split("*/")
55             new_word=to_remove[1]
56             if new_word:
57                 words_list_master.append(new_word)
58             continue
59
60         # storing the words
61         if comment_var==False:
62             if ":" in word:
63                 to_remove=word.split(":")
64                 new_word=to_remove[0]
65                 if new_word:
66                     words_list_master.append(new_word)
67                 words_list_master.append(":")
68                 new_word=to_remove[1]
69                 if new_word:
70                     words_list_master.append(new_word)
71                 continue
72
73             if "->" in word:
74                 to_remove=word.split("->")
75                 new_word=to_remove[0]
76                 if new_word:
77                     words_list_master.append(new_word)
78                 words_list_master.append("->")
79                 new_word=to_remove[1]
80                 if new_word:
81                     words_list_master.append(new_word)
82                 continue
83
84
85         words_list_master.append(word)
86
87 main_stack=[]

```

---

---

```

88 activities_list=[]
89 for word in words_list_master:
90     main_Stack.append(word)
91
92     if word=="}":
93         data_Stack=[]
94         main_Stack.pop()
95         while(True):
96             x=main_Stack.pop()
97             if x=="{":
98                 break
99             data_Stack.append(x)
100
101     # print(data_Stack)
102
103     if main_Stack[-1]=="actions":
104         if data_Stack:
105             temp_graph=nx.DiGraph()
106             rev_data_Stack=data_Stack[::-1]
107
108             for word2 in range(len(rev_data_Stack)):
109                 if rev_data_Stack[word2]==":":
110                     node_name=rev_data_Stack[word2-1]
111
112                     if rev_data_Stack[word2+1]=="claim":
113                         type_var="claim"
114                         resource_var=rev_data_Stack[word2+2]
115                     elif rev_data_Stack[word2+1]=="release":
116                         type_var="release"
117                         resource_var=rev_data_Stack[word2+2]
118                     elif rev_data_Stack[word2+1]=="move":
119                         type_var="action"
120                         resource_var=rev_data_Stack[word2+2].
121                             split(".")[0]
122                     else:
123                         type_var="action"
124                         resource_var=rev_data_Stack[word2+1].
125                             split(".")[0]
126
127                     temp_graph.add_node(node_name, resource=
128                         resource_var, type=type_var)
129                     empty_act=False
130                 else:
131                     empty_act=True
132                 # print(temp_graph.nodes)
133
134     if main_Stack[-1]=="flow":
135         if data_Stack:
136             rev_data_Stack=data_Stack[::-1]

```

---

---

```

134
135     for word2 in range(len(rev_data_Stack)-1):
136         if rev_data_Stack[word2]=="->":
137             if rev_data_Stack[word2+1].startswith("|"):
138                 sync_nodes=[]
139                 for word3 in range(len(rev_data_Stack)):
140                     if rev_data_Stack[word3]==
141                         rev_data_Stack[word2+1]:
142                         if word3<len(rev_data_Stack)-1:
143                             if rev_data_Stack[word3+1]=="
144                                 ->":
145                                 sync_nodes.append(
146                                     rev_data_Stack[word3
147                                         +2])
148                 for sn in sync_nodes:
149                     temp_graph.add_edge(rev_data_Stack[
150                         word2-1],sn)
151
152             else:
153                 if not rev_data_Stack[word2-1].startswith
154                     ("|"):
155                     temp_graph.add_edge(rev_data_Stack[
156                         word2-1],rev_data_Stack[word2+1])
157
158         empty_act=False
159     else:
160         empty_act=True
161
162     if main_Stack[-2]=="activity" and not empty_act:
163         activities_list.append([copy.deepcopy(main_Stack[-1]),
164             copy.deepcopy(temp_graph)])
165
166 activities_list_names=[element[0] for element in activities_list]
167
168 #End: Extraction of LSAT variables
169
170 size_of_seq=3 #max size of sequence
171
172 #Start timer
173 start=time.time()
174
175 activity_path='Example.ctrlsys_statespace.cif'
176 activityG=activitySequenceExtractor(activity_path)
177
178 extracted_seq_list_names=[]
179 for i in activityG.nodes:

```

---

---

```

175     # print(i)
176     # print(activityG.nodes[i])
177
178     if activityG.nodes[i]['initial']:
179         for j in activityG.nodes:
180             if activityG.nodes[j]['marked']:
181
182                 paths=list(nx.all_simple_edge_paths(activityG, i, j,
183                                                         cutoff=size_of_seq))
184
185                 for l in paths:
186                     seq_temp=[]
187                     for k in l:
188                         seq_temp.append(activityG[k[0]][k[1]][k[2]]['
189                                         name' ])
190
191                     extracted_seq_list_names.append(copy.deepcopy(
192                         seq_temp))
193
194 #Start: Initial List of sequences to visited
195
196 arr_temp=[]
197 new_arr=[]
198 for i in range(size_of_seq):
199     arr_temp=list(p for p in itertools.product(activities_list,
200                                               repeat=i+1))
201     for j in arr_temp:
202         new_arr.append(j)
203
204 arr_names=[]
205 for i in new_arr:
206     instance_list=[]
207     list_to_be_passed=[]
208     list_to_be_passed_names=[]
209     for j in i:
210         instance_list.append(j[0])
211         list_to_be_passed_names.append(j[0]+"_"+str(instance_list.
212                                                     count(j[0])))
213     arr_names.append(list_to_be_passed_names)
214
215 arr=[]
216 for i in arr_names:
217     temp_list=[]
218     for j in range(len(i)):
219
220         temp_graph1=copy.deepcopy(new_arr[arr_names.index(i)][j][1])

```

---

---

```

219     if i[j] not in imp_act:
220         nodes_list=list(temp_graph1.nodes)
221
222         for k in nodes_list:
223             if temp_graph1.nodes[k]['type']!= 'claim' and
                temp_graph1.nodes[k]['type']!= 'release':
224                 remove_node_pred=True
225                 remove_node_succ=True
226
227                 for n in temp_graph1.predecessors(k):
228                     if temp_graph1.nodes[n]['resource']!=
                        temp_graph1.nodes[k]['resource']:
229                         remove_node_pred=False
230                         break
231
232                 for n in temp_graph1.successors(k):
233                     if temp_graph1.nodes[n]['resource']!=
                        temp_graph1.nodes[k]['resource']:
234                         remove_node_succ=False
235                         break
236
237                 if remove_node_pred and remove_node_succ:
238                     pred_node=temp_graph1.predecessors(k)
239                     succ_node=temp_graph1.successors(k)
240                     temp_graph1.remove_node(k)
241                     elist=[]
242                     for pred in pred_node:
243                         for succ in succ_node:
244                             elist.append((pred,succ))
245
246                     temp_graph1.add_edges_from(elist)
247
248                 temp_list.append([i[j],copy.deepcopy(temp_graph1)])
249
250     else:
251         nodes_list=list(temp_graph1.nodes)
252
253         for k in nodes_list:
254             if temp_graph1.nodes[k]['type']!= 'claim' and
                temp_graph1.nodes[k]['type']!= 'release':
255                 if k not in imp_action[i[j]]:
256
257                     remove_node_pred=True
258                     remove_node_succ=True
259
260                     for n in temp_graph1.predecessors(k):
261                         if temp_graph1.nodes[n]['resource']!=
                            temp_graph1[k]['resource']:
262                             remove_node_pred=False

```

---

---

```

263         break
264
265     for n in temp_graph1.successors(k):
266         if temp_graph1.nodes[n]['resource']!=
                temp_graph1[k]['resource']:
267             remove_node_succ=False
268             break
269
270     if remove_node_pred and remove_node_succ:
271         pred_node=temp_graph1.predecessors(k)
272         succ_node=temp_graph1.successors(k)
273         temp_graph1.remove_node(k)
274         elist=[]
275         for pred in pred_node:
276             for succ in succ_node:
277                 elist.append((pred,succ))
278
279         temp_graph1.add_edges_from(elist)
280
281     temp_list.append([i[j],copy.deepcopy(temp_graph1)])
282
283
284
285     arr.append(copy.deepcopy(temp_list))
286
287     print(len(arr))
288     #Taking intersection of sequences
289     arr_names_temp=copy.deepcopy(arr_names)
290     arr_temp=copy.deepcopy(arr)
291     arr_names=[]
292     arr=[]
293     for i in range(len(arr_names_temp)):
294         if arr_names_temp[i] in extracted_seq_list_names:
295             arr_names.append(arr_names_temp[i])
296             arr.append(arr_temp[i])
297
298     print(len(arr))
299     # End: Initial List of sequences to visited
300
301
302     #Start: Separating important and non-important activities
303
304     unimp_act=[]
305     for i in activities_list:
306         for j in range(size_of_seq):
307             if (i[0]+"_"+str(j+1)) not in imp_act:
308                 unimp_act.append(i[0]+"_"+str(j+1))
309
310

```

---



---

```

311 #End: Separating important and non-important activities
312
313
314 finalReq="requirement req:\n"
315 cntr=0
316 seq_accepted=[]
317 seq_accepted_names=[]
318 redundant_seq_names=[]
319 no_of_synths=0
320
321 for i in arr:
322     list_to_be_passed=i
323     list_to_be_passed_names=arr_names[arr.index(i)]
324
325     if list_to_be_passed_names not in redundant_seq_names and
326        list_to_be_passed_names not in seq_accepted_names:
327         Y=ConvertToAutomata(list_to_be_passed)
328         X=Y.stringtowrite()
329         fileDirPlant=os.path.join("F:/Personal/TU Eindhoven/Post
330             Registration/Grad Project/LSAT/se-software-commandline-win-win
331             -x64-r9682/bin/","temp..cif")
332         Z=""
333
334         Z_trav=Z.main.splitlines()
335         for line in Z_trav:
336             if "edge " in line:
337                 wrd_arr=line.split()
338                 act_to_remove=wrd_arr[wrd_arr.index("edge")+1].split(
339                     ".")[0]
340
341                 act_available=[item[0] for item in list_to_be_passed]
342                 if act_to_remove in act_available:
343                     Z=Z+line+"\n"
344                 else:
345                     if "edge " not in Z_trav[Z_trav.index(line)+1]
346                        and "initial;" not in Z_trav[Z_trav.index(line)
347                            +1] and "marked;" not in Z_trav[Z_trav.index(
348                            line)+1]:
349                         if "location " in Z_trav[Z_trav.index(line)
350                            -1]:
351                             pos=Z.rfind(':')
352                             Z=Z[:pos]+";"
353                     else:
354                         Z=Z+line+"\n"
355
356         X=X+"\n"+Z
357         f=open(fileDirPlant,"w")
358         f.write(X)

```

---

---

```

352     f.close()
353     # Supervisory synthesis process
354     synth = subprocess.run(
355         [ 'F:/Personal/TU Eindhoven/Post
           Registration/Grad Project/LSAT/se-
           software-cmdline-win-win-x64-r9682/bin
           /cif3datasynth.bat', fileDirPlant],
356         capture_output=True,
357         text=True
358         )
359     no_of_synths+=1
360     print("Sequence visited:"+str(list_to_be_passed_names))
361     if "ERROR" not in synth.stderr:
362         seq_accepted_names.append(list_to_be_passed_names)
363         for k in range(1, size_of_seq-len(list_to_be_passed_names)
364             +1):
365             x=[element for element in itertools.product(unimp_act
366                 ,repeat=k)]
367             for l in x:
368                 y=list(itertools.chain(list_to_be_passed_names,l)
369                     )
370                 if y in arr_names:
371                     seq_accepted_names.append(y)
372                 y=list(itertools.chain(l,list_to_be_passed_names)
373                     )
374                 if y in arr_names:
375                     seq_accepted_names.append(y)
376             # print(seq_accepted_names)
377         for k in range(arr_names.index(list_to_be_passed_names)
378             +1,len(arr_names)):
379             if subseqchecker(arr_names[k],
380                 list_to_be_passed_names) and arr_names[k] not in
381                 seq_accepted_names:
382                 if all(item in unimp_act for item in diff(
383                     arr_names[k],list_to_be_passed_names)):
384                     seq_accepted_names.append(arr_names[k])
385                 # print("Potential addition :")
386                 # print(arr_names[k])
387     else:
388         print(synth.stderr+"in: "+str(arr.index(i)))
389         for k in range(1, size_of_seq-len(list_to_be_passed_names)
390             +1):
391             x=[element for element in itertools.product(unimp_act
392                 ,repeat=k)]

```

---

---

```

388         for l in x:
389             y=list(itertools.chain(list_to_be_passed_names , l)
390                               )
391             redundant_seq_names.append(y)
392             y=list(itertools.chain(l , list_to_be_passed_names)
393                               )
394             redundant_seq_names.append(y)
395
396
397 finalReq=finalReq+"\tlocation L0:\n\t\tinitial;\n"
398 cntr=1
399
400 for i in [item[0] for item in seq_accepted_names]:
401     finalReq=finalReq+"\t\tedge "+i+" goto LSeq"+str(cntr)+"_1;\n"
402     cntr+=1
403
404 cntr=1
405 for i in seq_accepted_names:
406     for j in i:
407         if i.index(j)>0:
408             finalReq=finalReq+"\tlocation LSeq"+str(cntr)+"_"
409             +str(i.index(j))+":\n"
410             finalReq=finalReq+"\t\tedge "+j+" goto LSeq"+str(
411                 cntr)+"_"+str(i.index(j)+1)+";\n"
412
413             finalReq=finalReq+"\tlocation LSeq"+str(cntr)+"_"
414             +str(i.index(j)+1)+":\n\t\tmarked;\n\n"
415             cntr+=1
416
417
418 finalReq=finalReq+"\nend"
419 print(finalReq)
420 fileDirfinal=os.path.join("F:/Personal/TU Eindhoven/Post Registration
421 /Grad Project/LSAT/se-software-commandline-win-win-x64-r9682/bin/" ,
422 "req_final_new_3.cif")
423 fin=open(fileDirfinal , "w")
424 fin.write(finalReq)
425 fin.close()
426 print("Total synths:"+str(no_of_synths))
427
428 #End timer
429 end=time.time()
430
431 print("Time:")
432 print(end-start)

```

---

---

## J Auxiliary code: subseqchecker

```
1 import networkx as nx
2 from pathlib import Path
3
4
5 #Larger array comes first
6 def subseqchecker(arr1, arr2):
7     ind=-1
8     for i in arr2:
9         if i in arr1:
10            if arr1.index(i)<=ind:
11                return False
12            else:
13                ind=arr1.index(i)
14        else:
15            return False
16
17    if ind===-1:
18        return False
19    else:
20        return True
21
22 #Smaller array comes first
23 def subseqcheckerend(arr1, arr2):
24     if arr1[0] not in arr2:
25         return False
26     else:
27         part=arr2[arr2.index(arr1[0]):arr2.index(arr1[0])+len(arr1)]
28         if part==arr1:
29             return True
30         else:
31             return False
32
33
34
35 def diff(arr1, arr2):
36     return list(set(arr1) - set(arr2)) + list(set(arr2) - set(arr1))
37
38
39 def writetofile(seq_accepted_names):
40     finalReq2=""
41     finalReq=""
42     seq_accepted_name=seq_accepted_names[0]
43     cntr=seq_accepted_names[1]
44     unimp_act=seq_accepted_names[2]
45     for j in seq_accepted_name:
46         if seq_accepted_name.index(j)==0:
47             finalReq=finalReq+"\t\tedge "+j+" goto LSeq"+str(cntr)+"
```

---

```

        _l;\n"
48     if seq_accepted_name.index(j)>0:
49         finalReq2=finalReq2+"\tlocation LSeq"+str(cntr)+"_"+str(
            seq_accepted_name.index(j))+":\n"
50         finalReq2=finalReq2+"\t\tedge "+j+" goto LSeq"+str(cntr)+
            "_"+str(seq_accepted_name.index(j)+1)+";\n"
51
52     finalReq2=finalReq2+"\tlocation LSeq"+str(cntr)+"_"+str(
            seq_accepted_name.index(j)+1)+":\n\t\tmarked;\n"
53     for k in unimp_act:
54         if k not in seq_accepted_name:
55             finalReq2=finalReq2+"\t\tedge "+k+";\n"
56
57     return [finalReq, finalReq2]
58
59
60 def automataToAutomataDAG(Z_main):
61     automataDAG=nx.DiGraph()
62     Z_trav=Z_main.splitlines()
63     for line in Z_trav:
64         if "location " in line:
65             node_temp=line.split()[1]
66             node_temp=node_temp.split(":")[0]
67             if node_temp not in list(automataDAG.nodes):
68                 automataDAG.add_node(node_temp, initial=False, marked=
                    False)
69             else:
70                 automataDAG.nodes[node_temp]['initial']=False
71                 automataDAG.nodes[node_temp]['marked']=False
72
73         if "initial;" in line:
74             automataDAG.nodes[node_temp]['initial']=True
75
76         if "marked;" in line:
77             automataDAG.nodes[node_temp]['marked']=True
78
79         if "edge " in line:
80             edges_temp=line.split()
81             edge_name=edges_temp[edges_temp.index('edge')+1]
82             if 'goto' not in edges_temp:
83                 dest_node=node_temp
84             else:
85                 dest_node_temp=edges_temp[edges_temp.index('goto')+1]
86                 dest_node=dest_node_temp.split(":")[0]
87
88             if (node_temp, dest_node) not in automataDAG.edges:
89                 automataDAG.add_edge(node_temp, dest_node, edgeName=[
                    edge_name])
90         else:

```

---

---

```
91         automataDAG.edges[node_temp, dest_node][ 'edgeName' ].
           append(edge_name)
92
93     initial_nodes=[]
94     marked_nodes=[]
95     for node_ in list(automataDAG.nodes):
96         if automataDAG.nodes[node_][ 'initial' ]:
97             initial_nodes.append(node_)
98
99         if automataDAG.nodes[node_][ 'marked' ]:
100             marked_nodes.append(node_)
101
102     all_paths=[]
103     for source in initial_nodes:
104         for dest in marked_nodes:
105             if source!=dest:
106                 paths= list(nx.all_simple_edge_paths(automataDAG,
107                                                         source, dest))
107                 for path in paths:
108                     print(path)
109     return automataDAG
```

---

## K Auxiliary code: ConvertToAutomata

```
1
2 class ConvertToAutomata:
3     #Initialize class instances
4     def __init__(self, activities_dict):
5         self.activities_dict=activities_dict
6
7
8     def checkcommon(self, list1, list2):
9         for x in list1:
10            for y in list2:
11                if x==y:
12                    return True
13
14            return False
15
16     #function to create string to be written to file
17     def stringtowrite(self):
18         string_x=""
19
20         for i in self.activities_dict:
21
22             #Writing activity names to string
23             string_x=string_x+"\n"
24
25
26             string_x=string_x+"plant "+i[0]+": "
27
28             #Writing edge names to string
29             string_x=string_x+"\n"
30             for j in i[1].nodes:
31                 string_x=string_x+"\tuncontrollable "+j+";\n"
32
33             #Writing state transitions to string
34             #dictionary that holds the locations and edges
35             string_x=string_x+"\n"
36             automatalocs={}
37
38             #list that holds locations to visit
39             listoflocs=[]
40             listoflocs.append(list(i[1].nodes))
41
42             for j in listoflocs:
43
44                 locname=''.join(map(str, j))
45                 automatalocs[locname]=[]
46
47                 for k in j:
```





---

```

91         automatalocs_claim[i[1].nodes[j]]['resource'
92             ].append([i[0],j])
93     else:
94         automatalocs_claim[i[1].nodes[j]]['resource'
95             ]=[i[0],j]
96     if i[1].nodes[j]['type']=='release':
97         if i[1].nodes[j]['resource'] in automatalocs_rel:
98             automatalocs_rel[i[1].nodes[j]]['resource'].
99                 append([i[0],j])
100     else:
101         automatalocs_rel[i[1].nodes[j]]['resource'
102             ]=[i[0],j]
103
104 for i in automatalocs_claim:
105     string_x=string_x+"\n"
106     string_x=string_x+"plant availability_"+i+":\n"
107     string_x=string_x+"\tlocation unclaimed:\n\t\tinitial;\n\t\tmarked;\n"
108     temp3=automatalocs_claim[i][:]
109     res = [i for n, i in enumerate(temp3) if i not in temp3[:
110         n]]
111     for j in res:
112         string_x=string_x+"\t\tedge "+j[0]+". "+j[1]+" goto
113             claimed;\n"
114
115     string_x=string_x+"\tlocation claimed:\n"
116     temp3=automatalocs_rel[i][:]
117     res = [i for n, i in enumerate(temp3) if i not in temp3[:
118         n]]
119     for j in res:
120         string_x=string_x+"\t\tedge "+j[0]+". "+j[1]+" goto
121             unclaimed;\n"
122
123     string_x=string_x+"end\n"
124
125 for i in automatalocs_claim:
126     cntr=0
127     string_x=string_x+"\n"
128     string_x=string_x+"plant claimingAutomata_"+i+":\n"
129     for j in automatalocs_claim[i]:
130         if cntr==0:
131             string_x=string_x+"\tlocation 1"+str(cntr)+":\n\t\t\tinitial;\n"
132         else:
133             string_x=string_x+"\tlocation 1"+str(cntr)+":\n"
134
135     cntr+=1
136     string_x=string_x+"\t\tedge "+j[0]+". "+j[1]+" goto 1"

```

---



---

**L Declaration: TU/e Code of Scientific Conduct**

## Declaration concerning the TU/e Code of Scientific Conduct for the Master's thesis

I have read the TU/e Code of Scientific Conduct<sup>i</sup>.

I hereby declare that my Master's thesis has been carried out in accordance with the rules of the TU/e Code of Scientific Conduct

Date

27/09/2021  
.....

Name

SAIKAT CHAKRABORTY  
.....

ID-number

1413961  
.....

Signature

Sai kat Chakraborty .  
.....

*Submit the signed declaration to the student administration of your department.*

<sup>i</sup> See: <http://www.tue.nl/en/university/about-the-university/integrity/scientific-integrity/>

The Netherlands Code of Conduct for Academic Practice of the VSNU can be found here also.  
More information about scientific integrity is published on the websites of TU/e and VSNU