MASTER

Interoperability between LSAT and CIF

Chakraborty, Saikat

Award date:
2021

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

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™)[1]. 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.

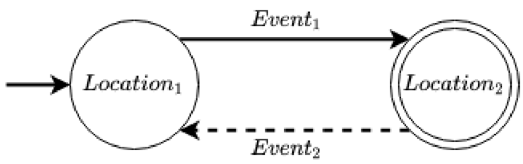


Figure 1: Automata example

[1]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.

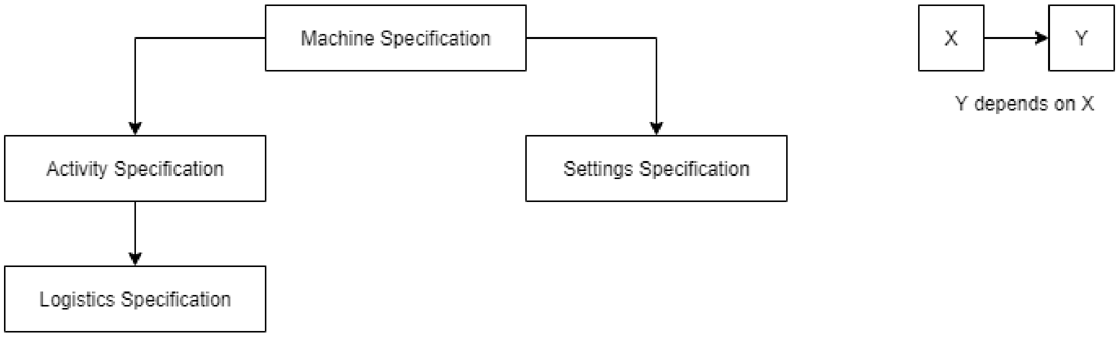


Figure 2: Dependencies between DSLs

- **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

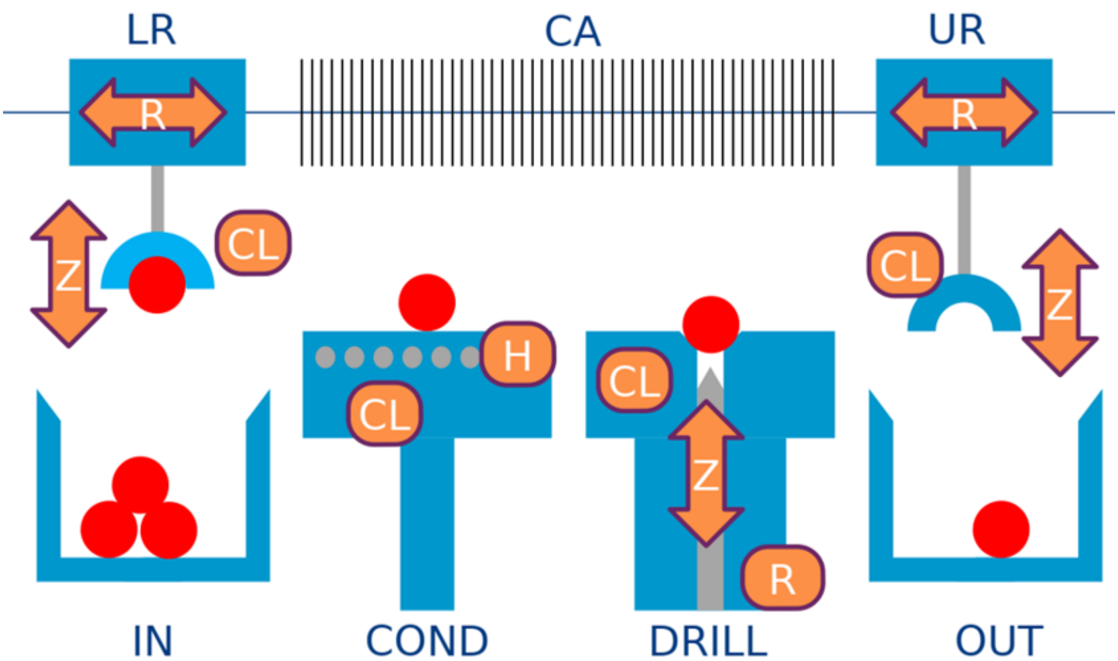


Figure 3: Twilight system. Adapted from [8]

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 ω 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

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

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 ω is selected from Seq. The activity, claiming and availability automata for ω 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 $l = 3$: $\omega = Act^1_A; Act^1_B; Act^2_A$. Then, the automata shown in Fig 5. describes the original behaviour of the system (plant) to which requirements are to be added.

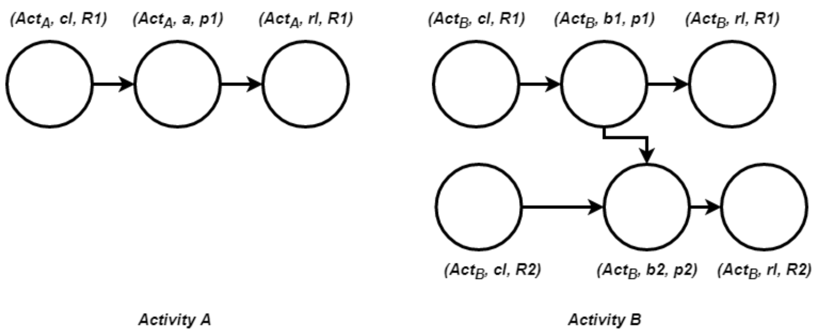
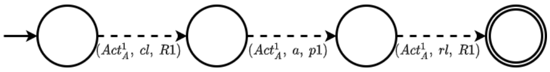
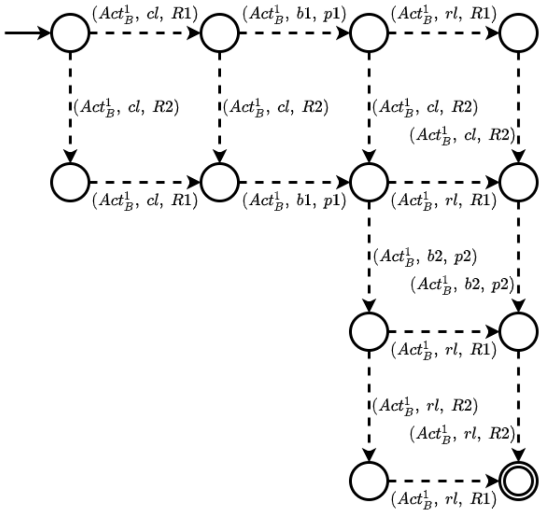


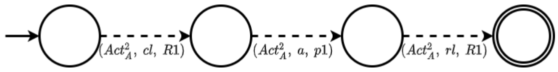
Figure 4: Example Act



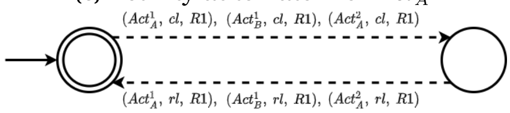
(a) Activity automaton for Act_A^1



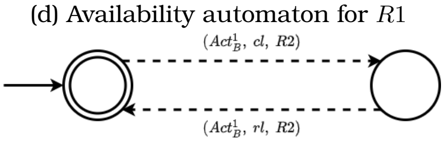
(b) Activity automaton for Act_B^1



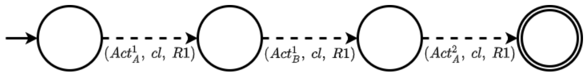
(c) Activity automaton for Act_A^2



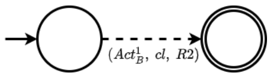
(d) Availability automaton for $R1$



(e) Availability automaton for $R2$



(f) Claiming automaton for $R1$



(g) Claiming automaton for $R2$

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 ω 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 ω, 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 $\omega_1 = Act^1_A; Act^1_B$ and $\omega_2 = Act^1_A; Act^1_C; Act^1_B$. 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.

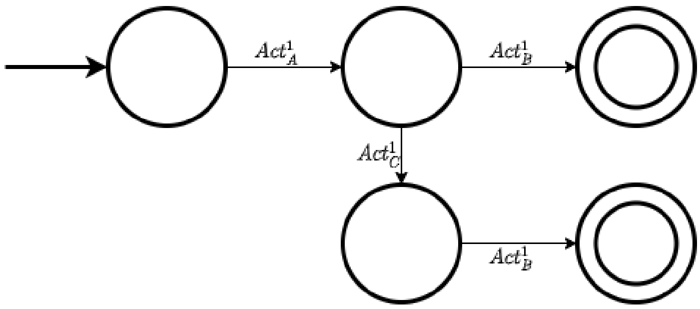


Figure 6: Example requirement automata to synthesize supervisor for LSAT

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

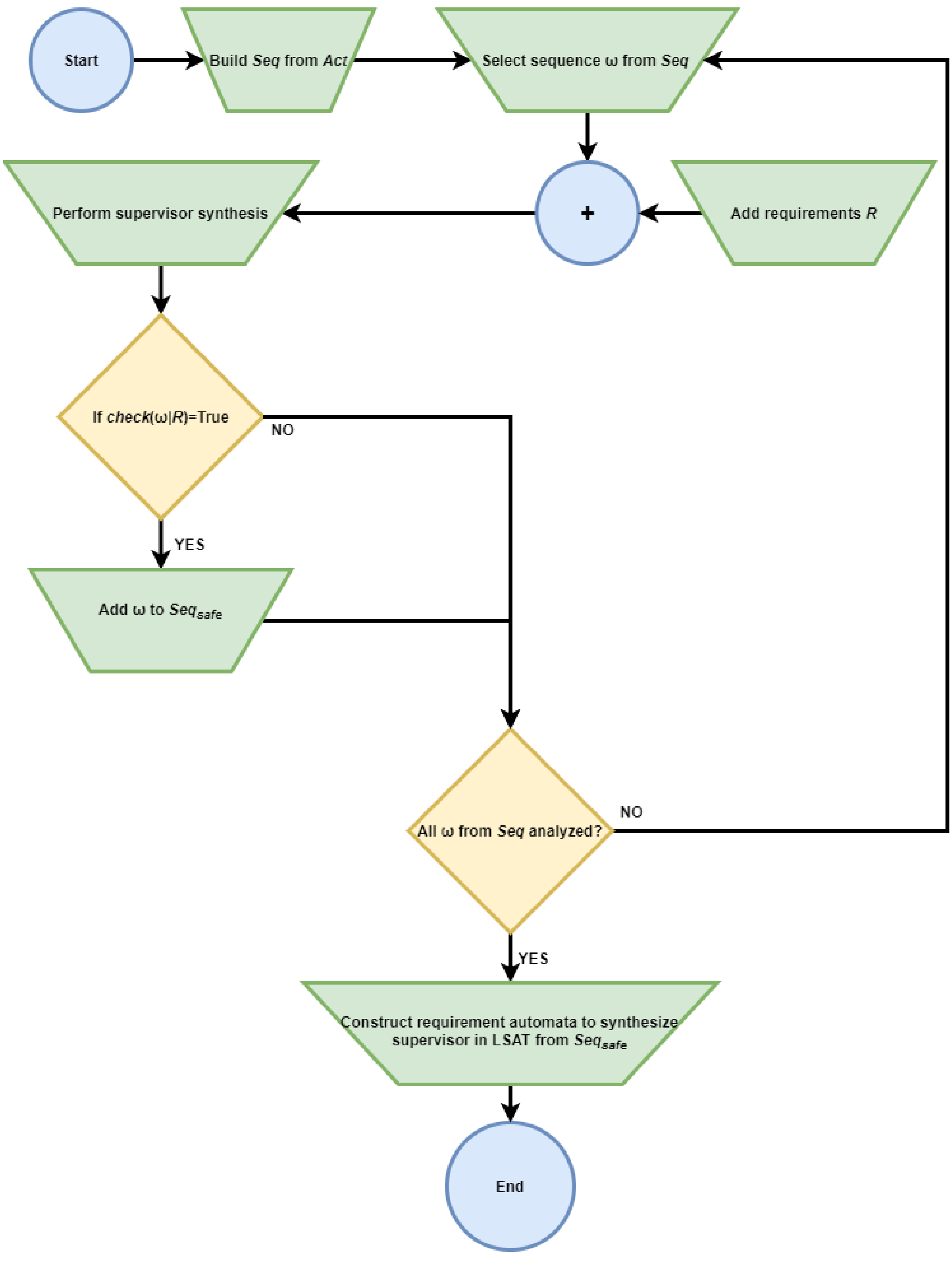


Figure 7: Process flowchart: Method I

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 ω 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 ω_1, ω_2 contain only unimportant activities. Similarly, given a sequence ω 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 ω_1, ω_2 are empty sequences or contain only unimportant activities.

   This is because ω_1, ω_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 ω

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 ω_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 ω. 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 ω, 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.

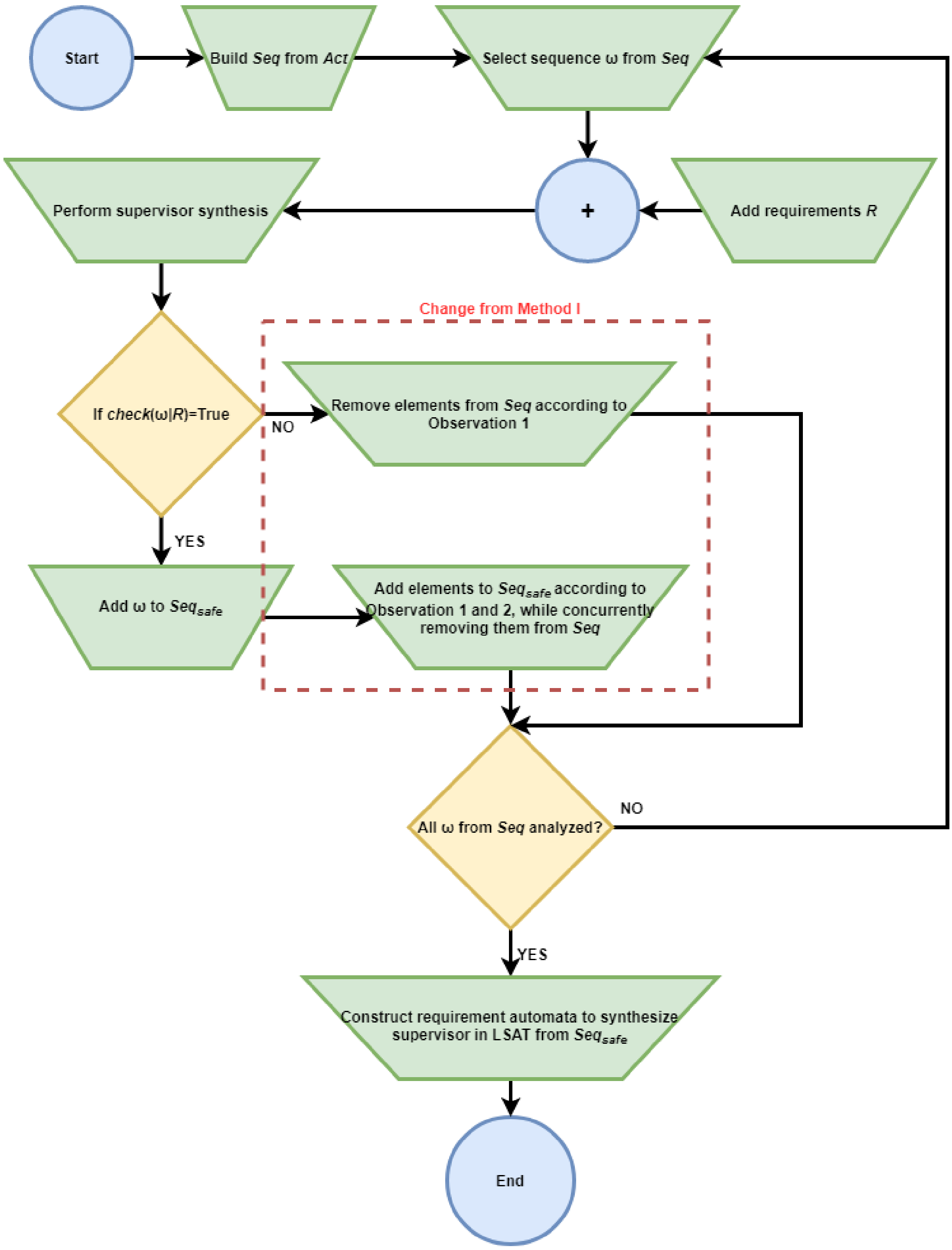


Figure 8: Process flowchart: Method II

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 ω 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 ω, 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.

(a) DAG of activity before abstraction

(b) DAG of activity after abstraction

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.

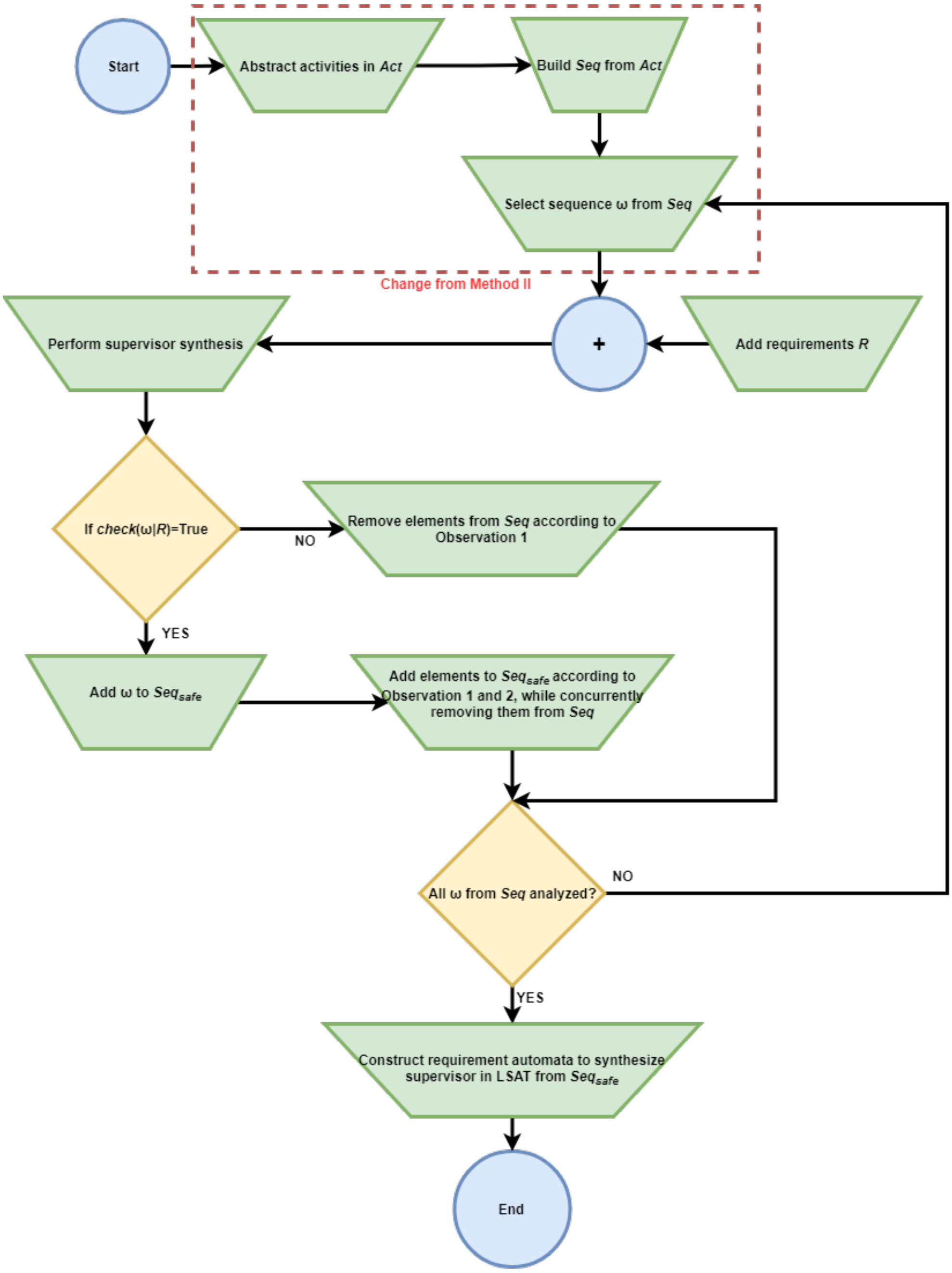


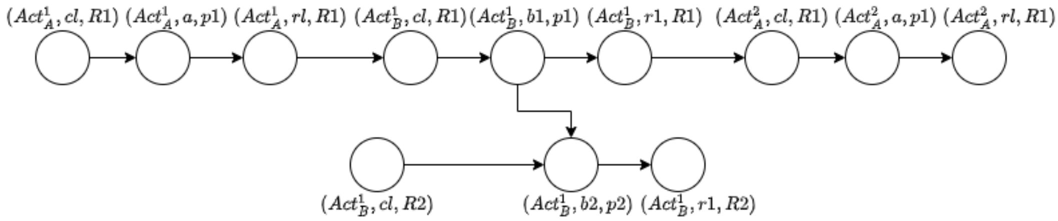
Figure 10: Process flowchart: Method III

2.4 Method IV

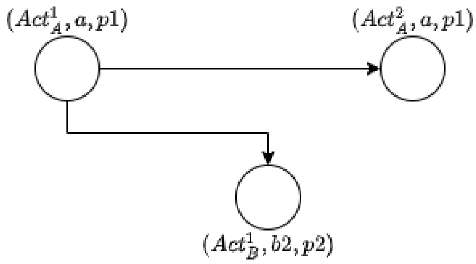
This method uses the principles of abstraction similar to the one used in Method III. However, this method deviates slightly from the step of setting up of activity, claiming and availability automata used thus far in Methods I, II, and III.

Instead of making activity automata for each constituent activity of a sequence ω and then using the claim automata to describe the sequencing of the activities in the sequence, the combined DAG of the sequence ω 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 ω. 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.



(a) Sequence-DAG before abstraction



(b) Sequence-DAG after abstraction

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.

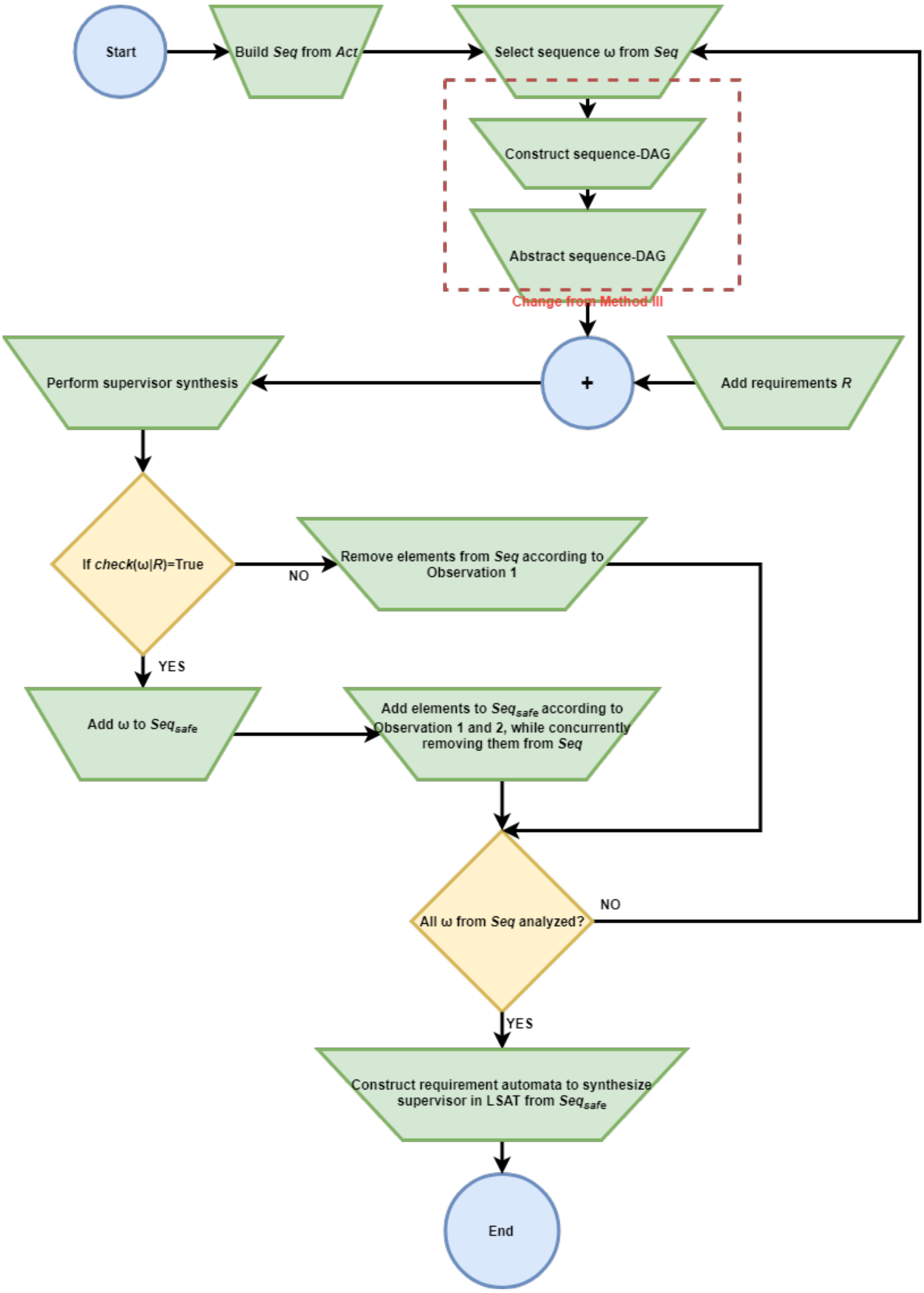


Figure 12: Process flowchart: Method IV

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 Ω_{np} is the set containing all sequences ω_{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.

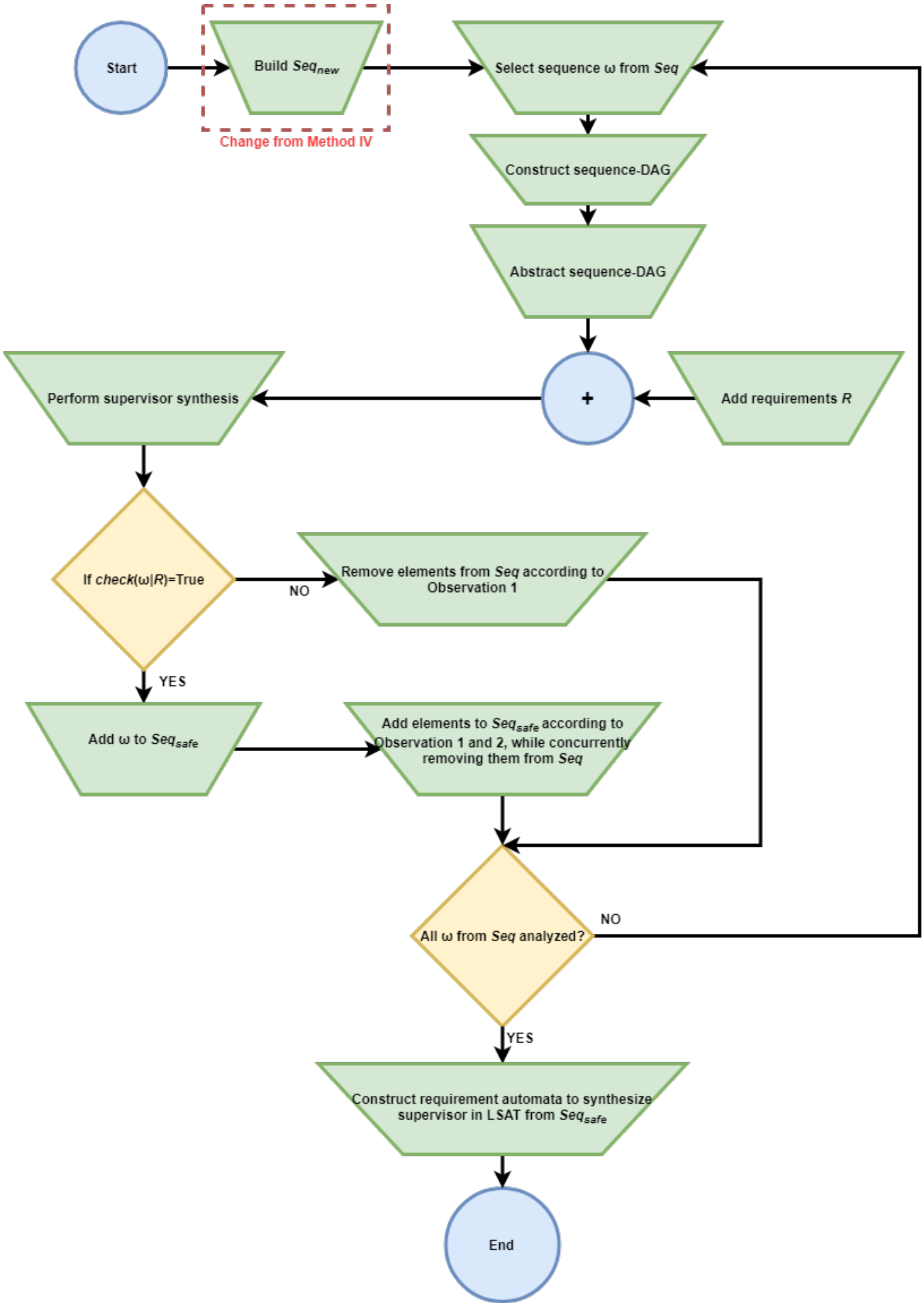


Figure 13: Process flowchart: Method V

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

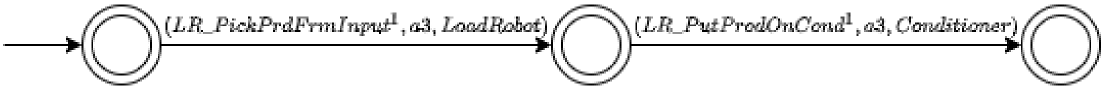
For Method V, the requirement for the activities in the sequences is as shown in Fig. 15 and is mentioned below:

- 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

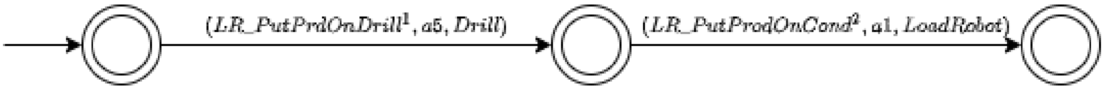
The codes for the various methods are detailed in Appendix E-I. All snippets have been commented for clarity.

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.



(a) Requirement 1



(b) Requirement 2

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

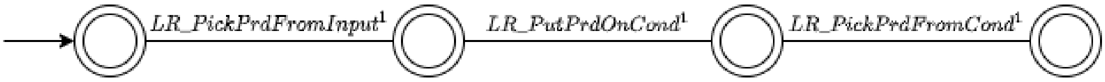


Figure 15: Automata describing the defined requirements between the action instances

- **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,

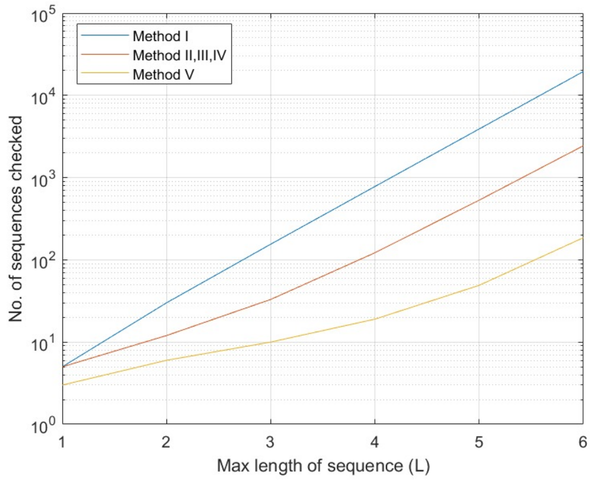


Figure 16: No. of sequences checked vs max length of Sequence ($\mathcal{L}$)

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

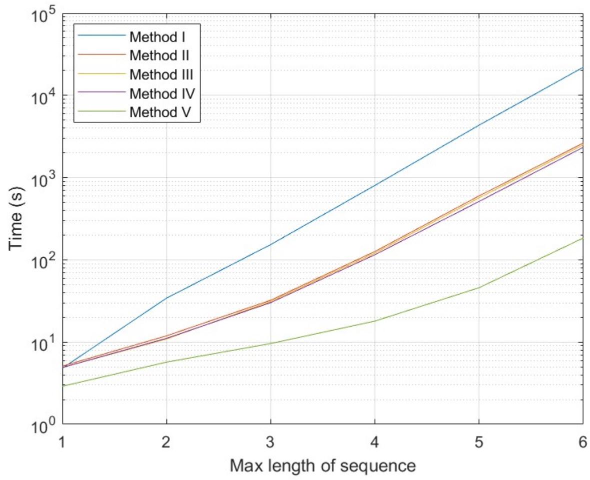


Figure 17: Time (s) vs max length of Sequence ($\mathcal{L}$)

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

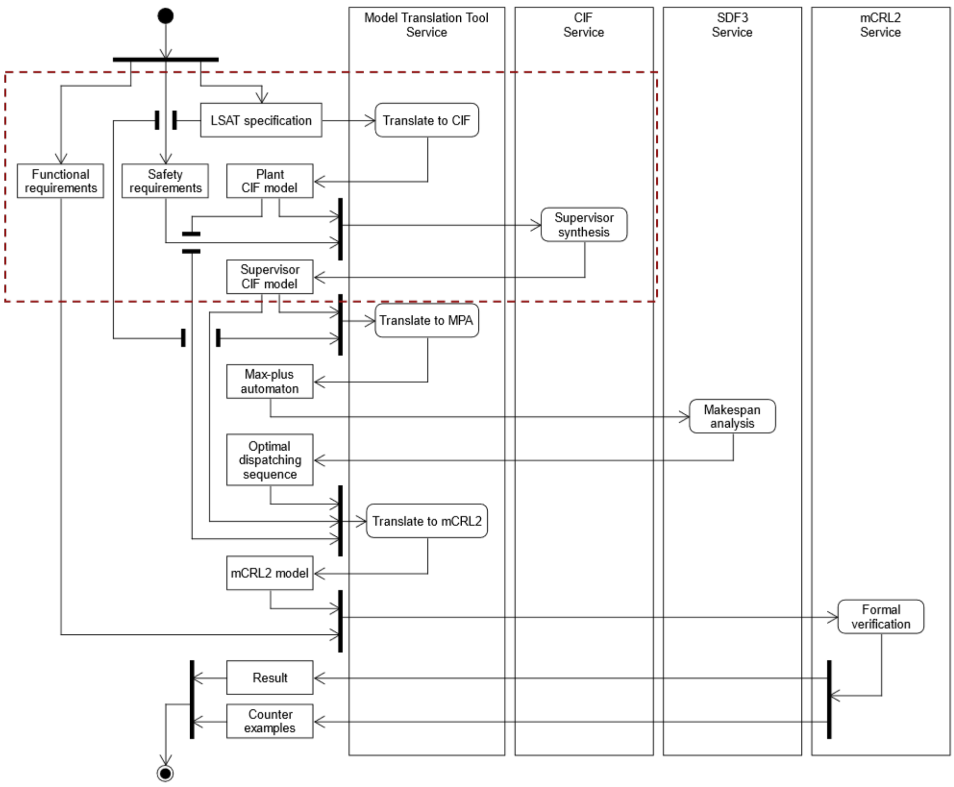


Figure 18: Arrowhead Framework example

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

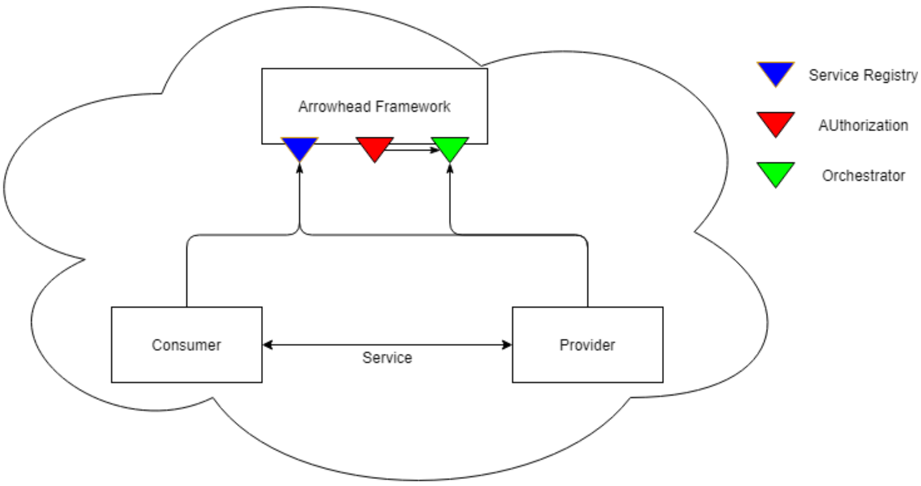


Figure 19: Arrowhead Framework example

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.

[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". Nederlands. 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.

[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.

[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 nº 826452 (project Arrowhead Tools) and from the partners national programs/funding authorities.

Appendices

A LSAT code: Machine specification for Twilight system

```
Machine Twilight

PeripheralType Clamp {
  Actions {
    clamp
    unclamp
  }
}

PeripheralType XYMotor {
  SetPoints {
    X [m]
    Y [m]
  }
  Axes {
    X [m] moves X
    Y [m] moves Y
  }
}

PeripheralType DrillMotor {
        Actions {
                on
                off
        }
}

PeripheralType Drill {
  SetPoints {
    Z [m]
  }
  Axes {
    Z [mm] moves Z
  }
  Conversion "Z=Z/1000"
}

PeripheralType Conditioner {
  Actions {
    condition
  }
}
```

```
Resource Drill {
  CL: Clamp
  DL: DrillMotor
  ZR: Drill {
    SymbolicPositions {
      UP
      DOWN
    }
    Profiles (normal, slow)
    Paths {
      DOWN --> UP profile slow
      UP --> DOWN profile normal
    }
  }
}

Resource Conditioner {
  CL: Clamp
  CD: Conditioner
}

Resource LoadRobot {
  CL: Clamp
  XY: XYMotor {
    AxisPositions {
      X (IN, COND, DRILL)
      Y (ABOVE, AT)
    }
    SymbolicPositions {
      ABOVE_IN (X.IN, Y.ABOVE)
      ABOVE_COND (X.COND, Y.ABOVE)
      ABOVE_DRILL (X.DRILL, Y.ABOVE)
      AT_IN (X.IN, Y.AT)
      AT_COND (X.COND, Y.AT)
      AT_DRILL (X.DRILL, Y.AT)
      OUT_DRILL (X.DRILL)
    }
    Profiles (normal)
    Paths {
      FullMesh {
        profile normal
        ABOVE_IN
        ABOVE_COND
        ABOVE_DRILL
      }
      ABOVE_IN <-> AT_IN profile normal
      ABOVE_COND <-> AT_COND profile normal
      ABOVE_DRILL <-> AT_DRILL profile normal
```

```
      ABOVE_DRILL <-> OUT_DRILL profile normal
      OUT_DRILL <-> AT_DRILL profile normal
    }
  }
}

Resource UnloadRobot {
  CL: Clamp
  XY: XYMotor {
    AxisPositions {
      X (COND, DRILL, OUT)
      Y (ABOVE, AT)
    }
    SymbolicPositions {
      ABOVE_OUT (X.OUT, Y.ABOVE)
      ABOVE_COND (X.COND, Y.ABOVE)
      ABOVE_DRILL (X.DRILL, Y.ABOVE)
      AT_OUT (X.OUT, Y.AT)
      AT_COND (X.COND, Y.AT)
      AT_DRILL (X.DRILL, Y.AT)
      OUT_DRILL (X.DRILL)
    }
    Profiles (normal)
    Paths {
      FullMesh {
        profile normal
        ABOVE_COND
        ABOVE_DRILL
        ABOVE_OUT
      }
      ABOVE_COND <-> AT_COND profile normal
      ABOVE_DRILL <-> AT_DRILL profile normal
      ABOVE_DRILL <-> OUT_DRILL profile normal
      OUT_DRILL <-> AT_DRILL profile normal
      ABOVE_OUT <-> AT_OUT profile normal
    }
  }
}
```

B LSAT code: Settings specification for Twilight system

```
import "twilight.machine"

LoadRobot.CL {
  Timings {
    clamp = Pert(min=0.1, max=1, mode=0.250, gamma=10)
    unclamp = 0.200
  }
}

LoadRobot.XY {
  Axis X {
    Profiles {
      normal (V = 1, A = 8, J = 20)
    }
    Positions {
      IN = 1
      COND = 2
      DRILL = 3
    }
  }
  Axis Y {
    Profiles {
      normal (V = 2, A = 15, J = 35)
    }
    Positions {
      ABOVE = 0
      OUT_DRILL = 0.8
      AT = 2
    }
  }
}

UnloadRobot.CL {
  Timings {
    clamp = 0.25

    unclamp = 0.200
  }
}

UnloadRobot.XY {
  Axis X {
    Profiles {
      normal (V = 8, A = 8, J = 20)
    }
    Positions {
```

```
      COND = 2
      DRILL = 3
      OUT = 4
    }
  }
  Axis Y {
    Profiles {
      normal (V = 15, A = 15, J = 35)
    }
    Positions {
      ABOVE = 0
      OUT_DRILL = 0.8
      AT = 2
    }
  }
}

Conditioner.CL {
  Timings {
    clamp = 0.250
    unclamp = 0.200
  }
}

Conditioner.CD {
  Timings {
    condition = 5.0
  }
}

Drill.CL {
  Timings {
    clamp = 0.250
    unclamp = 0.200
  }
}

Drill.DL{
        Timings {
    on = 0.5
    off = 0.5
        }
}

Drill.ZR {
  Axis Z {
    Profiles {
      normal (V = 0.1, A = 1, J = 5)
      slow (V = 0.1, A = 0.1, J = 1.0)
```

```
    }
    Positions {
      UP = 100
      DOWN = 0
    }
  }
}
```

C LSAT code: Activity specification for Twilight system

```
import "twilight.machine"


activity LR_PickPrdFromInput {
  prerequisites {
    LoadRobot.XY at ABOVE_IN
  }
  actions {
    C1: claim LoadRobot
    R1: release LoadRobot
    A1: move LoadRobot.XY to AT_IN with speed profile normal
    A2: LoadRobot.CL.clamp
    A3: move LoadRobot.XY to ABOVE_IN with speed profile normal
  }
  action flow {
    C1 -> A1 -> A2 -> A3 -> R1
  }
}

activity LR_PutPrdOnCond {
  prerequisites {
    LoadRobot.XY at ABOVE_IN
  }
  actions {
    C1: claim LoadRobot
    C2: claim Conditioner
    R1: release LoadRobot
    R2: release Conditioner
    C3: claim CollisionArea
    R3: release CollisionArea
    A1: move LoadRobot.XY to ABOVE_COND with speed profile normal
    A2: move LoadRobot.XY to AT_COND with speed profile normal
    A3: Conditioner.CL.clamp
    A4: LoadRobot.CL.unclamp
    A5: move LoadRobot.XY to ABOVE_COND with speed profile normal
    A6: move LoadRobot.XY to ABOVE_IN with speed profile normal
  }
  action flow {
    C1 -> C3 -> A1 -> A2 -> C2 -> A3 -> A4 -> R2 -> A5 -> A6 ->  R3
        ->R1
  }
}

activity LR_PutPrdOnDrill {
  prerequisites {
    LoadRobot.XY at ABOVE_IN
```

```
  }
  actions {
    C1: claim LoadRobot
    C2: claim Drill
    R1: release LoadRobot
    R2: release Drill
    C3: claim CollisionArea
    R3: release CollisionArea
    A2: move LoadRobot.XY to ABOVE_DRILL with speed profile normal
    A3: move LoadRobot.XY to OUT_DRILL with speed profile normal
    A4: move LoadRobot.XY to AT_DRILL with speed profile normal
    A5: Drill.CL.clamp
    A6: LoadRobot.CL.unclamp
    A7: move LoadRobot.XY to OUT_DRILL with speed profile normal
    A8: move LoadRobot.XY to ABOVE_DRILL with speed profile normal
    A10: move LoadRobot.XY to ABOVE_IN with speed profile normal
  }
  action flow {
    C1 -> C3  -> A2 -> A3 -> A4 -> C2 -> A5 -> A6 -> A7 -> R2 -> A8 -> A10  -> R3-> R1
  }
}

activity LR_PickPrdFromCond {
  prerequisites {
    LoadRobot.XY at ABOVE_IN
  }
  actions {
    C1: claim LoadRobot
    C2: claim Conditioner
    R1: release LoadRobot
    R2: release Conditioner
    C3: claim CollisionArea
    R3: release CollisionArea
    A1: move LoadRobot.XY to ABOVE_COND with speed profile normal
    A2: move LoadRobot.XY to AT_COND with speed profile normal
    A3: Conditioner.CL.unclamp
    A4: LoadRobot.CL.clamp
    A5: move LoadRobot.XY to ABOVE_COND with speed profile normal
    A6: move LoadRobot.XY to ABOVE_IN with speed profile normal
  }
  action flow {
    C1 -> C3 -> A1 -> A2 -> C2 -> A4 -> A3 -> R2 -> A5 -> A6  -> R3-> R1
  }
}

activity LR_PickPrdFromDrill {
  prerequisites {
```

```
    LoadRobot.XY at ABOVE_IN
  }
  actions {
    C1: claim LoadRobot
    C2: claim Drill
    R1: release LoadRobot
    R2: release Drill
    C3: claim CollisionArea
    R3: release CollisionArea
    A1: move LoadRobot.XY to ABOVE_COND with speed profile normal
    A2: move LoadRobot.XY to ABOVE_DRILL with speed profile normal
    A3: move LoadRobot.XY to OUT_DRILL with speed profile normal
    A4: move LoadRobot.XY to AT_DRILL with speed profile normal
    A5: Drill.CL.unclamp
    A6: LoadRobot.CL.clamp
    A7: move LoadRobot.XY to OUT_DRILL with speed profile normal
    A8: move LoadRobot.XY to ABOVE_DRILL with speed profile normal
    A9: move LoadRobot.XY to ABOVE_COND with speed profile normal
    A10: move LoadRobot.XY to ABOVE_IN with speed profile normal
  }
  action flow {
    C1 -> C3 ->  A1 -> A2 -> A3 -> A4 -> C2 -> A6 -> A5 -> R2 -> A7
        -> A8 -> A9 -> A10  -> R3-> R1
  }
}
```

D CIF code: Requirements

```
requirement req1:
        location l0:
                initial;
                marked;
                edge LR_PickPrdFromInput_1.A3 goto l1;
        location l1:
                marked;
                edge LR_PutPrdOnCond_1.A3 goto l2;
        location l2:
                marked;
end

requirement req2:
        location l0:
                initial;
                marked;
                edge LR_PutPrdOnDrill_1.A5 goto l1;
        location l1:
                marked;
                edge LR_PutPrdOnCond_2.A1 goto l2;
        location l2:
                marked;
end
```

E Python code: Method I

```
#Method1: Crude Method where all possible sequences are individually
    checked

import networkx as nx
from itertools import product
import subprocess
import copy
import time
from pathlib import Path
from ConvertToAutomata import ConvertToAutomata
from subseqchecker import writetofile


fileDirReq=Path("se-software-cmdline-win-win-x64-r9682/bin/
    re_final_report.cif")
req=open(fileDirReq,"r")
Z_main=req.read()
req.close()

#Start: Separating important and non-important activities

Z_trav=Z_main.splitlines()
imp_act=[]
for line in Z_trav:
    if "edge " in line:
        wrd_arr=line.split()
        imp_act.append(wrd_arr[wrd_arr.index("edge")+1].split(".")
            [0])

#Start: Extraction of LSAT variables

words_list_master=[]
comment_var=False
with open('twilight.activity','r') as file:

    # reading each line
    for line in file:

        # reading each word
        for word in line.split():

            #Remove comments
            if word.startswith("//"):
                break

            if word.startswith("/*"):
                comment_var=True
```

```
            if "*/" in word:
                comment_var=False
                to_remove=word.split("*/")
                new_word=to_remove[1]
                if new_word:
                    words_list_master.append(new_word)
                continue

            # storing the words
            if comment_var==False:
                if ":" in word:
                    to_remove=word.split(":")
                    new_word=to_remove[0]
                    if new_word:
                        words_list_master.append(new_word)
                    words_list_master.append(":")
                    new_word=to_remove[1]
                    if new_word:
                        words_list_master.append(new_word)
                    continue

                if "->" in word:
                    to_remove=word.split("->")
                    new_word=to_remove[0]
                    if new_word:
                        words_list_master.append(new_word)
                    words_list_master.append("->")
                    new_word=to_remove[1]
                    if new_word:
                        words_list_master.append(new_word)
                    continue


                words_list_master.append(word)

main_Stack=[]
activities_list=[]
for word in words_list_master:
    main_Stack.append(word)

    if word=="}":
        data_Stack=[]
        main_Stack.pop()
        while(True):
            x=main_Stack.pop()
            if x=="{":
                break
            data_Stack.append(x)
```

```

        # print(data_Stack)

        if main_Stack[-1]=="actions":
            if data_Stack:
                temp_graph=nx.DiGraph()
                rev_data_Stack=data_Stack[::-1]

                for word2 in range(len(rev_data_Stack)):
                    if rev_data_Stack[word2]==":":
                        node_name=rev_data_Stack[word2-1]

                        if rev_data_Stack[word2+1]=="claim":
                            type_var="claim"
                            resource_var=rev_data_Stack[word2+2]
                        elif rev_data_Stack[word2+1]=="release":
                            type_var="release"
                            resource_var=rev_data_Stack[word2+2]
                        elif rev_data_Stack[word2+1]=="move":
                            type_var="action"
                            resource_var=rev_data_Stack[word2+2].
                                split(".")[0]
                        else:
                            type_var="action"
                            resource_var=rev_data_Stack[word2+1].
                                split(".")[0]

                        temp_graph.add_node(node_name,resource=
                            resource_var,type=type_var)
                empty_act=False
            else:
                empty_act=True
            # print(temp_graph.nodes)

        if main_Stack[-1]=="flow":
            if data_Stack:
                rev_data_Stack=data_Stack[::-1]

                for word2 in range(len(rev_data_Stack)-1):
                    if rev_data_Stack[word2]=="->":
                        if rev_data_Stack[word2+1].startswith("|"):
                            sync_nodes=[]
                            for word3 in range(len(rev_data_Stack)):
                                if rev_data_Stack[word3]==
                                    rev_data_Stack[word2+1]:
                                    if word3<len(rev_data_Stack)-1:
                                        if rev_data_Stack[word3+1]=="
                                            ->":
                                            sync_nodes.append(
```

```
                                        rev_data_Stack[word3+2])
                            for sn in sync_nodes:
                                temp_graph.add_edge(rev_data_Stack[word2-1],sn)


                        else:
                            if not rev_data_Stack[word2-1].startswith("|"):
                                temp_graph.add_edge(rev_data_Stack[word2-1],rev_data_Stack[word2+1])

                empty_act=False
            else:
                empty_act=True

        if main_Stack[-2]=="activity" and not empty_act:
            activities_list.append([copy.deepcopy(main_Stack[-1]),copy.deepcopy(temp_graph)])


activities_list_names=[element[0] for element in activities_list]
#End: Extraction of LSAT variables

synths_array=[]
time_var=[]

size_of_seq=1 #Max size of sequence

#Start: Initial List of sequences to visited
arr_temp=[]

new_arr_names_only=[]
for i in range(size_of_seq):
    arr_temp=list(p for p in product(activities_list_names, repeat=i+1))
    new_arr_names_only=new_arr_names_only+arr_temp

arr_names=[]

for i in new_arr_names_only:
    instance_list=[]
    list_to_be_passed=[]
    list_to_be_passed_names=[]
    for j in i:
        instance_list.append(j)
        list_to_be_passed_names.append(j+"_"+str(instance_list.count(j)))
```

```python

    arr_names.append(list_to_be_passed_names)

# End: Initial List of sequences to visited


unimp_act=[]
for i in activities_list:
    for j in range(size_of_seq):
        if (i[0]+"_"+str(j+1)) not in imp_act:
            unimp_act.append(i[0]+"_"+str(j+1))


    #End: Separating important and non-important activities
finalReq="requirement req:\n"
cntr=0
seq_accepted_names=[]
no_of_synths=0
synthPath=Path('se-software-cmdline-win-win-x64-r9682/bin/cif3datasynth.bat')
start=time.time()
for i in arr_names:
    list_to_be_passed_names=i
    list_to_be_passed=[]

    for j in range(len(i)):
        list_to_be_passed.append([i[j],activities_list[activities_list_names.index(new_arr_names_only[arr_names.index(i)][j])][1]])

    Y=ConvertToAutomata(list_to_be_passed)
    X=Y.stringtowrite()
    fileDirPlant=Path("se-software-cmdline-win-win-x64-r9682/bin/temp_.cif")
    Z=""
    Z_trav=Z_main.splitlines()
    for line in Z_trav:
        if "edge " in line:
            wrd_arr=line.split()
            act_to_remove=wrd_arr[wrd_arr.index("edge")+1].split(".")[0]

            act_available=[item[0] for item in list_to_be_passed]
            if act_to_remove in act_available:
                Z=Z+line+"\n"
            else:
                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]:
```

```
                    if "location " in Z_trav[Z_trav.index(line)-1]:
                        pos=Z.rfind(':')
                        Z=Z[:pos]+";"
        else:
            Z=Z+line+"\n"


    X=X+"\n"+Z
    f=open(fileDirPlant,"w")
    f.write(X)
    f.close()
    # Supervisory synthesis process
    synth = subprocess.run(
                        [synthPath.absolute().as_posix(), fileDirPlant.absolute().as_posix()],
                         capture_output=True,
                         text=True
                         )
    no_of_synths+=1
    print("Sequence visited: "+str(no_of_synths))
    if "ERROR" not in synth.stderr:
        seq_accepted_names.append(list_to_be_passed_names)


end=time.time()

print("Total synths:"+str(no_of_synths))
print("Total time:"+str(end-start))

finalReq=finalReq+"\tlocation L0:\n\t\tinitial;\n"
cntr=1

for i in [item[0] for item in seq_accepted_names]:
    finalReq=finalReq+"\t\tedge "+i+" goto LSeq"+str(cntr)+"_1;\n"
    cntr+=1

cntr=1
for i in seq_accepted_names:
            for j in i:
                if i.index(j)>0:
                    finalReq=finalReq+"\tlocation LSeq"+str(cntr)+"_"+str(i.index(j))+":\n"
                    finalReq=finalReq+"\t\tedge "+j+" goto LSeq"+str(cntr)+"_"+str(i.index(j)+1)+";\n"

            finalReq=finalReq+"\tlocation LSeq"+str(cntr)+"_"+str(i.index(j)+1)+":\n\t\tmarked;\n\n"
            cntr+=1

```

```
finalReq=finalReq+"\nend"

fileDirfinal=Path("se-software-cmdline-win-win-x64-r9682/bin/
    req_final_method_1.cif")
fin=open(fileDirfinal,"w")
fin.write(finalReq)
fin.close()
```

F Python code: Method II

```python
#Method 2: Improved method where sequences with addition of
#unimportant activities are not checked

import networkx as nx
import subprocess
import copy
import time
from pathlib import Path
from itertools import product
from subseqchecker import subseqchecker
from subseqchecker import diff
from subseqchecker import subseqcheckerend
from ConvertToAutomata import ConvertToAutomata


fileDirReq=Path("se-software-cmdline-win-win-x64-r9682/bin/
    re_final_report.cif")
req=open(fileDirReq,"r")
Z_main=req.read()
req.close()


#Start: Separating important and non-important activities

Z_trav=Z_main.splitlines()
imp_act=[]
for line in Z_trav:
    if "edge " in line:
        wrd_arr=line.split()
        imp_act.append(wrd_arr[wrd_arr.index("edge")+1].split(".")
            [0])

#Start: Extraction of LSAT variables

words_list_master=[]
comment_var=False
with open('twilight.activity','r') as file:

    # reading each line
    for line in file:

        # reading each word
        for word in line.split():

            #Remove comments
            if word.startswith("//"):
                break
```

```

            if word.startswith("/*"):
                comment_var=True

            if "*/" in word:
                comment_var=False
                to_remove=word.split("*/")
                new_word=to_remove[1]
                if new_word:
                    words_list_master.append(new_word)
                continue

            # storing the words
            if comment_var==False:
                if ":" in word:
                    to_remove=word.split(":")
                    new_word=to_remove[0]
                    if new_word:
                        words_list_master.append(new_word)
                    words_list_master.append(":")
                    new_word=to_remove[1]
                    if new_word:
                        words_list_master.append(new_word)
                    continue

                if "->" in word:
                    to_remove=word.split("->")
                    new_word=to_remove[0]
                    if new_word:
                        words_list_master.append(new_word)
                    words_list_master.append("->")
                    new_word=to_remove[1]
                    if new_word:
                        words_list_master.append(new_word)
                    continue


                words_list_master.append(word)

main_Stack=[]
activities_list=[]
for word in words_list_master:
    main_Stack.append(word)

    if word=="}":
        data_Stack=[]
        main_Stack.pop()
        while(True):
            x=main_Stack.pop()
```

```
            if x=="{":
                break
            data_Stack.append(x)

        # print(data_Stack)

        if main_Stack[-1]=="actions":
            if data_Stack:
                temp_graph=nx.DiGraph()
                rev_data_Stack=data_Stack[::-1]

                for word2 in range(len(rev_data_Stack)):
                    if rev_data_Stack[word2]==":":
                        node_name=rev_data_Stack[word2-1]

                        if rev_data_Stack[word2+1]=="claim":
                            type_var="claim"
                            resource_var=rev_data_Stack[word2+2]
                        elif rev_data_Stack[word2+1]=="release":
                            type_var="release"
                            resource_var=rev_data_Stack[word2+2]
                        elif rev_data_Stack[word2+1]=="move":
                            type_var="action"
                            resource_var=rev_data_Stack[word2+2].
                                split(".")[0]
                        else:
                            type_var="action"
                            resource_var=rev_data_Stack[word2+1].
                                split(".")[0]

                        temp_graph.add_node(node_name,resource=
                            resource_var,type=type_var)
                empty_act=False
            else:
                empty_act=True
            # print(temp_graph.nodes)

        if main_Stack[-1]=="flow":
            if data_Stack:
                rev_data_Stack=data_Stack[::-1]

                for word2 in range(len(rev_data_Stack)-1):
                    if rev_data_Stack[word2]=="->":
                        if rev_data_Stack[word2+1].startswith("|"):
                            sync_nodes=[]
                            for word3 in range(len(rev_data_Stack)):
                                if rev_data_Stack[word3]==
                                    rev_data_Stack[word2+1]:
                                    if word3<len(rev_data_Stack)-1:
```

```
                                        if rev_data_Stack[word3+1]=="->":
                                            sync_nodes.append(rev_data_Stack[word3+2])
                            for sn in sync_nodes:
                                temp_graph.add_edge(rev_data_Stack[word2-1],sn)


                        else:
                            if not rev_data_Stack[word2-1].startswith("|"):
                                temp_graph.add_edge(rev_data_Stack[word2-1],rev_data_Stack[word2+1])

                empty_act=False
            else:
                empty_act=True

        if main_Stack[-2]=="activity" and not empty_act:
            activities_list.append([copy.deepcopy(main_Stack[-1]), copy.deepcopy(temp_graph)])

activities_list_names=[element[0] for element in activities_list]

#End: Extraction of LSAT variables


size_of_seq=1 #max size of sequence

#Start: Initial List of sequences to visited

arr_temp=[]

new_arr_names_only=[]
for i in range(size_of_seq):
    arr_temp=list(p for p in product(activities_list_names, repeat=i+1))
    new_arr_names_only=new_arr_names_only+arr_temp

arr_names=[]

for i in new_arr_names_only:
    instance_list=[]
    list_to_be_passed=[]
    list_to_be_passed_names=[]
    for j in i:
        instance_list.append(j)
```

```
            list_to_be_passed_names.append(j+"_"+str(instance_list.count(j)))

        arr_names.append(list_to_be_passed_names)


# End: Initial List of sequences to visited

unimp_act=[]
for i in activities_list:
    for j in range(size_of_seq):
        if (i[0]+"_"+str(j+1)) not in imp_act:
            unimp_act.append(i[0]+"_"+str(j+1))


#End: Separating important and non-important activities
indexes_to_be_visited=list(range(len(arr_names)))
len_cntr=[]
for i in range(1,size_of_seq+1):
    len_cntr.append(pow(len(activities_list_names),i))

finalReq="requirement req:\n"
cntr=0
seq_accepted_names=[]
no_of_synths=0
synthPath=Path('se-software-cmdline-win-win-x64-r9682/bin/cif3datasynth.bat')
start=time.time()
for i in arr_names:
    list_to_be_passed_names=i

    if arr_names.index(i) in indexes_to_be_visited:
        indexes_to_be_visited.remove(arr_names.index(i))

        for r in len_cntr:
            if arr_names.index(i)<r:
                reg=r
                break

        list_to_be_passed=[]

        for j in range(len(i)):
            list_to_be_passed.append([i[j],activities_list[activities_list_names.index(new_arr_names_only[arr_names.index(i)][j])][1]])

        Y=ConvertToAutomata(list_to_be_passed)
        X=Y.stringtowrite()
        fileDirPlant=Path("se-software-cmdline-win-win-x64-r9682/bin/
```

```
            temp_.cif")
        Z=""

        Z_trav=Z_main.splitlines()
        for line in Z_trav:
            if "edge " in line:
                wrd_arr=line.split()
                act_to_remove=wrd_arr[wrd_arr.index("edge")+1].split(".")[0]

                act_available=[item[0] for item in list_to_be_passed]
                if act_to_remove in act_available:
                    Z=Z+line+"\n"
                else:
                    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]:
                        if "location " in Z_trav[Z_trav.index(line)-1]:
                            pos=Z.rfind(':')
                            Z=Z[:pos]+";"
            else:
                Z=Z+line+"\n"


        X=X+"\n"+Z
        f=open(fileDirPlant,"w")
        f.write(X)
        f.close()
        # Supervisory synthesis process
        synth = subprocess.run(
                            [synthPath.absolute().as_posix(), fileDirPlant.absolute().as_posix()],
                            capture_output=True,
                            shell=True,
                            text=True
                            )
        no_of_synths+=1
        print("Sequence visited:"+str(no_of_synths))
        if "ERROR" not in synth.stderr:
            seq_accepted_names.append(list_to_be_passed_names)
            indexes_temp=list(indexes_to_be_visited)
            if len(indexes_to_be_visited)==0:
                break
            for k in indexes_temp:
                if k>=reg:
                    if subseqchecker(arr_names[k], list_to_be_passed_names):
```

```
                        if set(diff(arr_names[k], list_to_be_passed_names)).issubset(set(unimp_act)):
                            seq_accepted_names.append(arr_names[k])
                            indexes_to_be_visited.remove(k)

            # print(seq_accepted_names)

        else:

            indexes_temp=list(indexes_to_be_visited)
            if len(indexes_to_be_visited)==0:
                break
            for k in indexes_temp:
                if k>=reg:
                    if subseqcheckerend(list_to_be_passed_names, arr_names[k]):
                        if set(diff(arr_names[k], list_to_be_passed_names)).issubset(set(unimp_act)):
                            indexes_to_be_visited.remove(k)

end=time.time()


print("Total synths:"+str(no_of_synths))
print("Total time:"+str(end-start))


finalReq=finalReq+"\tlocation L0:\n\t\tinitial;\n"
cntr=1

for i in [item[0] for item in seq_accepted_names]:
    finalReq=finalReq+"\t\tedge "+i+" goto LSeq"+str(cntr)+"_1;\n"
    cntr+=1

cntr=1
for i in seq_accepted_names:
            for j in i:
                if i.index(j)>0:
                    finalReq=finalReq+"\tlocation LSeq"+str(cntr)+"_"+str(i.index(j))+":\n"
                    finalReq=finalReq+"\t\tedge "+j+" goto LSeq"+str(cntr)+"_"+str(i.index(j)+1)+";\n"

            finalReq=finalReq+"\tlocation LSeq"+str(cntr)+"_"+str(i.index(j)+1)+":\n\t\tmarked;\n\n"
            cntr+=1

```

```
finalReq=finalReq+"\nend"

fileDirfinal=Path("se-software-cmdline-win-win-x64-r9682/bin/
    req_final_method_2.cif")
fin=open(fileDirfinal,"w")
fin.write(finalReq)
fin.close()
```

G Python code: Method III

```
#Method 3: Improved method where actions which are not claim release
#or connecting activities are ignored

import networkx as nx
import subprocess
import copy
import time
from pathlib import Path
from itertools import product
from ConvertToAutomata import ConvertToAutomata
from subseqchecker import subseqchecker
from subseqchecker import subseqcheckerend
from subseqchecker import diff

fileDirReq=Path("se-software-cmdline-win-win-x64-r9682/bin/re_final_report.cif")
req=open(fileDirReq,"r")
Z_main=req.read()
req.close()

#Start: Separating important and non-important activities
Z_trav=Z_main.splitlines()
imp_act=[]
imp_action={}

for line in Z_trav:
    if "edge " in line:
        wrd_arr=line.split()
        imp_act.append(wrd_arr[wrd_arr.index("edge")+1].split(".")[0])
        if wrd_arr[wrd_arr.index("edge")+1].split(".")[0] in imp_action:
            imp_action[wrd_arr[wrd_arr.index("edge")+1].split(".")[0]].append( wrd_arr[wrd_arr.index("edge")+1].split(".")[1])
        else:
            imp_action[wrd_arr[wrd_arr.index("edge")+1].split(".")[0]]=[]
            imp_action[wrd_arr[wrd_arr.index("edge")+1].split(".")[0]].append( wrd_arr[wrd_arr.index("edge")+1].split(".")[1])

#Start: Extraction of LSAT variables

words_list_master=[]
comment_var=False
with open('twilight.activity','r') as file:
```

```python

    # reading each line
    for line in file:

        # reading each word
        for word in line.split():

            #Remove comments
            if word.startswith("//"):
                break

            if word.startswith("/*"):
                comment_var=True

            if "*/" in word:
                comment_var=False
                to_remove=word.split("*/")
                new_word=to_remove[1]
                if new_word:
                    words_list_master.append(new_word)
                continue

            # storing the words
            if comment_var==False:
                if ":" in word:
                    to_remove=word.split(":")
                    new_word=to_remove[0]
                    if new_word:
                        words_list_master.append(new_word)
                    words_list_master.append(":")
                    new_word=to_remove[1]
                    if new_word:
                        words_list_master.append(new_word)
                    continue

                if "->" in word:
                    to_remove=word.split("->")
                    new_word=to_remove[0]
                    if new_word:
                        words_list_master.append(new_word)
                    words_list_master.append("->")
                    new_word=to_remove[1]
                    if new_word:
                        words_list_master.append(new_word)
                    continue


                words_list_master.append(word)

```

```
main_Stack=[]
activities_list=[]
for word in words_list_master:
    main_Stack.append(word)

    if word=="}":
        data_Stack=[]
        main_Stack.pop()
        while(True):
            x=main_Stack.pop()
            if x=="{":
                break
            data_Stack.append(x)

        # print(data_Stack)

        if main_Stack[-1]=="actions":
            if data_Stack:
                temp_graph=nx.DiGraph()
                rev_data_Stack=data_Stack[::-1]

                for word2 in range(len(rev_data_Stack)):
                    if rev_data_Stack[word2]==":":
                        node_name=rev_data_Stack[word2-1]

                        if rev_data_Stack[word2+1]=="claim":
                            type_var="claim"
                            resource_var=rev_data_Stack[word2+2]
                        elif rev_data_Stack[word2+1]=="release":
                            type_var="release"
                            resource_var=rev_data_Stack[word2+2]
                        elif rev_data_Stack[word2+1]=="move":
                            type_var="action"
                            resource_var=rev_data_Stack[word2+2].
                                split(".")[0]
                        else:
                            type_var="action"
                            resource_var=rev_data_Stack[word2+1].
                                split(".")[0]

                        temp_graph.add_node(node_name,name=node_name,
                            resource=resource_var,type=type_var)
                empty_act=False
            else:
                empty_act=True
            # print(temp_graph.nodes)

        if main_Stack[-1]=="flow":
            if data_Stack:
```

```
                rev_data_Stack=data_Stack[::-1]

                for word2 in range(len(rev_data_Stack)-1):
                    if rev_data_Stack[word2]=="->":
                        if rev_data_Stack[word2+1].startswith("|"):
                            sync_nodes=[]
                            for word3 in range(len(rev_data_Stack)):
                                if rev_data_Stack[word3]==rev_data_Stack[word2+1]:
                                    if word3<len(rev_data_Stack)-1:
                                        if rev_data_Stack[word3+1]=="->":
                                            sync_nodes.append(rev_data_Stack[word3+2])
                            for sn in sync_nodes:
                                temp_graph.add_edge(rev_data_Stack[word2-1],sn)


                        else:
                            if not rev_data_Stack[word2-1].startswith("|"):
                                temp_graph.add_edge(rev_data_Stack[word2-1],rev_data_Stack[word2+1])

                empty_act=False
            else:
                empty_act=True

        if main_Stack[-2]=="activity" and not empty_act:
            activities_list.append([copy.deepcopy(main_Stack[-1]),copy.deepcopy(temp_graph)])


activities_list_names=[element[0] for element in activities_list]

#End: Extraction of LSAT variables

alternate_dag_imp=[]
alternate_dag_unimp=[]
for i in activities_list_names:
    match_found=False
    for j in imp_act:
        temp_name=copy.deepcopy(j.rsplit('_',1)[0])
        if temp_name==i:
            temp_graph1=copy.deepcopy(activities_list[activities_list_names.index(i)][1])
            nodes_list=list(temp_graph1.nodes)
```

```
            for k in nodes_list:
                if temp_graph1.nodes[k]['type']!='claim' and temp_graph1.nodes[k]['type']!='release':
                    if k not in imp_action[j]:

                        pred_node=list(temp_graph1.predecessors(k))
                        succ_node=list(temp_graph1.successors(k))
                        temp_graph1.remove_node(k)
                        elist=[]
                        for pred in pred_node:
                            for succ in succ_node:
                                elist.append((pred,succ))

                        temp_graph1.add_edges_from(elist)

            alternate_dag_imp.append(copy.deepcopy(temp_graph1))
            match_found=True
            break

    if not match_found:
        alternate_dag_imp.append(-1)

for i in activities_list_names:
    temp_graph1=copy.deepcopy(activities_list[activities_list_names.index(i)][1])
    nodes_list=list(temp_graph1.nodes)

    for k in nodes_list:
        if temp_graph1.nodes[k]['type']!='claim' and temp_graph1.nodes[k]['type']!='release':

            pred_node=list(temp_graph1.predecessors(k))
            succ_node=list(temp_graph1.successors(k))
            temp_graph1.remove_node(k)
            elist=[]
            for pred in pred_node:
                for succ in succ_node:
                    elist.append((pred,succ))

            temp_graph1.add_edges_from(elist)

    alternate_dag_unimp.append(copy.deepcopy(temp_graph1))

size_of_seq=6 #max size of sequence


#Start: Initial List of sequences to visited

```

```python
arr_temp=[]

new_arr_names_only=[]
for i in range(size_of_seq):
    arr_temp=list(p for p in product(activities_list_names, repeat=i+1))
    new_arr_names_only=new_arr_names_only+arr_temp

arr_names=[]

for i in new_arr_names_only:
    instance_list=[]
    list_to_be_passed=[]
    list_to_be_passed_names=[]
    for j in i:
        instance_list.append(j)
        list_to_be_passed_names.append(j+"_"+str(instance_list.count(j)))

    arr_names.append(list_to_be_passed_names)

# End: Initial List of sequences to visited

#Start: Separating important and non-important activities

unimp_act=[]
for i in activities_list:
    for j in range(size_of_seq):
        if (i[0]+"_"+str(j+1)) not in imp_act:
            unimp_act.append(i[0]+"_"+str(j+1))


#End: Separating important and non-important activities
indexes_to_be_visited=list(range(len(arr_names)))
len_cntr=[]
for i in range(1,size_of_seq+1):
    len_cntr.append(pow(len(activities_list_names),i))


finalReq="requirement req:\n"
cntr=0
seq_accepted_names=[]

no_of_synths=0
synthPath=Path('se-software-cmdline-win-win-x64-r9682/bin/cif3datasynth.bat')
start=time.time()
cntr=1
for i in arr_names:
```

```
list_to_be_passed_names=i

if arr_names.index(i) in indexes_to_be_visited:
    indexes_to_be_visited.remove(arr_names.index(i))
    for r in len_cntr:
        if arr_names.index(i)<r:
            reg=r
            break

    list_to_be_passed=[]

    for j in range(len(i)):
        if i[j] in imp_act:
            temp_graph=copy.deepcopy(activities_list[
                activities_list_names.index(new_arr_names_only[
                arr_names.index(i)][j])][1])
            for k in list(temp_graph.nodes):
                if temp_graph.nodes[k]['type']!='claim' and
                    temp_graph.nodes[k]['type']!='release':
                    if k not in imp_action[i[j]]:

                        pred_node=list(temp_graph.predecessors(k)
                            )
                        succ_node=list(temp_graph.successors(k))
                        temp_graph.remove_node(k)
                        elist=[]
                        for pred in pred_node:
                            for succ in succ_node:
                                elist.append((pred,succ))

                        temp_graph.add_edges_from(elist)

            list_to_be_passed.append([i[j],copy.deepcopy(
                temp_graph)])
        else:
            list_to_be_passed.append([i[j],activities_list[
                activities_list_names.index(new_arr_names_only[
                arr_names.index(i)][j])][1]])


    Y=ConvertToAutomata(list_to_be_passed)
    X=Y.stringtowrite()
    fileDirPlant=Path("se-software-cmdline-win-win-x64-r9682/bin/
        temp_new_3.cif")
    Z=""

    Z_trav=Z_main.splitlines()
    for line in Z_trav:
        if "edge " in line:
```

```
                wrd_arr=line.split()
                act_to_remove=wrd_arr[wrd_arr.index("edge")+1].split(".")[0]

                act_available=i
                if act_to_remove in act_available:
                    Z=Z+line+"\n"
                else:
                    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]:
                        if "location " in Z_trav[Z_trav.index(line)-1]:
                            pos=Z.rfind(':')
                            Z=Z[:pos]+";"
            else:
                Z=Z+line+"\n"


        X=X+"\n"+Z
        f=open(fileDirPlant,"w")
        f.write(X)
        f.close()
        # Supervisory synthesis process
        synth = subprocess.run(
                                    [synthPath.absolute().as_posix(), fileDirPlant.absolute().as_posix()],
                                    capture_output=True,
                                    text=True
                                    )
        no_of_synths+=1
        print("Sequence visited:"+str(no_of_synths))
        if "ERROR" not in synth.stderr:
            seq_accepted_names.append(list_to_be_passed_names)
            indexes_temp=list(indexes_to_be_visited)
            if len(indexes_to_be_visited)==0:
                break
            for k in indexes_temp:
                if k>=reg:
                    if subseqchecker(arr_names[k], list_to_be_passed_names):
                        if set(diff(arr_names[k], list_to_be_passed_names)).issubset(set(unimp_act)):
                            seq_accepted_names.append(arr_names[k])
                            indexes_to_be_visited.remove(k)

            # print(seq_accepted_names)
```

```
        else:
            indexes_temp=list(indexes_to_be_visited)
            if len(indexes_to_be_visited)==0:
                break
            for k in indexes_temp:
                if k>=reg:
                    if subseqcheckerend(list_to_be_passed_names,arr_names[k]):
                        if set(diff(arr_names[k],list_to_be_passed_names)).issubset(set(unimp_act)):
                            indexes_to_be_visited.remove(k)




end=time.time()


print("Total synths:"+str(no_of_synths))
print("Total time:"+str(end-start))


finalReq=finalReq+"\tlocation L0:\n\t\tinitial;\n"
cntr=1

for i in [item[0] for item in seq_accepted_names]:
    finalReq=finalReq+"\t\tedge "+i+" goto LSeq"+str(cntr)+"_1;\n"
    cntr+=1

cntr=1
for i in seq_accepted_names:
    for j in i:
        if i.index(j)>0:
            finalReq=finalReq+"\tlocation LSeq"+str(cntr)+"_"+str(i.index(j))+":\n"
            finalReq=finalReq+"\t\tedge "+j+" goto LSeq"+str(cntr)+"_"+str(i.index(j)+1)+";\n"

    finalReq=finalReq+"\tlocation LSeq"+str(cntr)+"_"+str(i.index(j)+1)+":\n\t\tmarked;\n\n"
    cntr+=1



finalReq=finalReq+"\nend"

fileDirfinal=Path("se-software-cmdline-win-win-x64-r9682/bin/
```

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

H Python code: Method IV

```python
import networkx as nx
import subprocess
import copy
import time
from multiprocessing import Pool
from pathlib import Path
from itertools import product
from ConvertToAutomata2 import ConvertToAutomata
from subseqchecker import subseqchecker
from subseqchecker import subseqcheckerend
from subseqchecker import diff
from subseqchecker import writetofile

if __name__=='__main__':
    fileDirReq=Path("se-software-cmdline-win-win-x64-r9682/bin/re_final_report.cif")
    req=open(fileDirReq,"r")
    Z_main=req.read()
    req.close()

    #Start: Separating important and non-important activities
    Z_trav=Z_main.splitlines()
    imp_act=[]
    imp_action={}

    for line in Z_trav:
        if "edge " in line:
            wrd_arr=line.split()
            imp_act.append(wrd_arr[wrd_arr.index("edge")+1].split(".")[0])
            if wrd_arr[wrd_arr.index("edge")+1].split(".")[0] in imp_action:
                imp_action[wrd_arr[wrd_arr.index("edge")+1].split(".")[0]].append( wrd_arr[wrd_arr.index("edge")+1].split(".")[1])
            else:
                imp_action[wrd_arr[wrd_arr.index("edge")+1].split(".")[0]]=[]
                imp_action[wrd_arr[wrd_arr.index("edge")+1].split(".")[0]].append( wrd_arr[wrd_arr.index("edge")+1].split(".")[1])

    #Start: Extraction of LSAT variables
    print(imp_action)
    words_list_master=[]
    comment_var=False
    with open('twilight.activity','r') as file:
```

```

        # reading each line
        for line in file:

            # reading each word
            for word in line.split():

                #Remove comments
                if word.startswith("//"):
                    break

                if word.startswith("/*"):
                    comment_var=True

                if "*/" in word:
                    comment_var=False
                    to_remove=word.split("*/")
                    new_word=to_remove[1]
                    if new_word:
                        words_list_master.append(new_word)
                    continue

                # storing the words
                if comment_var==False:
                    if ":" in word:
                        to_remove=word.split(":")
                        new_word=to_remove[0]
                        if new_word:
                            words_list_master.append(new_word)
                        words_list_master.append(":")
                        new_word=to_remove[1]
                        if new_word:
                            words_list_master.append(new_word)
                        continue

                    if "->" in word:
                        to_remove=word.split("->")
                        new_word=to_remove[0]
                        if new_word:
                            words_list_master.append(new_word)
                        words_list_master.append("->")
                        new_word=to_remove[1]
                        if new_word:
                            words_list_master.append(new_word)
                        continue


                    words_list_master.append(word)

```

```
    main_Stack=[]
    activities_list=[]
    for word in words_list_master:
        main_Stack.append(word)

        if word=="}":
            data_Stack=[]
            main_Stack.pop()
            while(True):
                x=main_Stack.pop()
                if x=="{":
                    break
                data_Stack.append(x)

            # print(data_Stack)

            if main_Stack[-1]=="actions":
                if data_Stack:
                    temp_graph=nx.DiGraph()
                    rev_data_Stack=data_Stack[::-1]

                    for word2 in range(len(rev_data_Stack)):
                        if rev_data_Stack[word2]==":":
                            node_name=rev_data_Stack[word2-1]

                            if rev_data_Stack[word2+1]=="claim":
                                type_var="claim"
                                resource_var=rev_data_Stack[word2+2]
                            elif rev_data_Stack[word2+1]=="release":
                                type_var="release"
                                resource_var=rev_data_Stack[word2+2]
                            elif rev_data_Stack[word2+1]=="move":
                                type_var="action"
                                resource_var=rev_data_Stack[word2+2].split(".")[0]
                            else:
                                type_var="action"
                                resource_var=rev_data_Stack[word2+1].split(".")[0]

                            temp_graph.add_node(node_name,name=node_name,resource=resource_var,type=type_var)
                    empty_act=False
                else:
                    empty_act=True
                # print(temp_graph.nodes)

            if main_Stack[-1]=="flow":
```

```
                if data_Stack:
                    rev_data_Stack=data_Stack[::-1]

                    for word2 in range(len(rev_data_Stack)-1):
                        if rev_data_Stack[word2]=="->":
                            if rev_data_Stack[word2+1].startswith("|"):
                                sync_nodes=[]
                                for word3 in range(len(rev_data_Stack)):
                                    if rev_data_Stack[word3]==rev_data_Stack[word2+1]:
                                        if word3<len(rev_data_Stack)-1:
                                            if rev_data_Stack[word3+1]=="->":
                                                sync_nodes.append(rev_data_Stack[word3+2])
                                for sn in sync_nodes:
                                    temp_graph.add_edge(rev_data_Stack[word2-1],sn)


                            else:
                                if not rev_data_Stack[word2-1].startswith("|"):
                                    temp_graph.add_edge(rev_data_Stack[word2-1],rev_data_Stack[word2+1])

                    empty_act=False
                else:
                    empty_act=True

            if main_Stack[-2]=="activity" and not empty_act:
                activities_list.append([copy.deepcopy(main_Stack[-1]),copy.deepcopy(temp_graph)])


    activities_list_names=[element[0] for element in activities_list]

    #End: Extraction of LSAT variables

    alternate_dag_imp=[]
    alternate_dag_unimp=[]
    for i in activities_list_names:
        match_found=False
        for j in imp_act:
```

```
            temp_name=copy.deepcopy(j.rsplit('_',1)[0])
            if temp_name==i:
                temp_graph1=copy.deepcopy(activities_list[activities_list_names.index(i)][1])
                nodes_list=list(temp_graph1.nodes)

                for k in nodes_list:
                    if temp_graph1.nodes[k]['type']!='claim' and temp_graph1.nodes[k]['type']!='release':

                        if k not in imp_action[j]:

                            pred_node=list(temp_graph1.predecessors(k))
                            succ_node=list(temp_graph1.successors(k))
                            temp_graph1.remove_node(k)
                            elist=[]
                            for pred in pred_node:
                                for succ in succ_node:
                                    elist.append((pred,succ))

                            temp_graph1.add_edges_from(elist)

                alternate_dag_imp.append(copy.deepcopy(temp_graph1))
                match_found=True
                break

        if not match_found:
            alternate_dag_imp.append(-1)

    for i in activities_list_names:
        temp_graph1=copy.deepcopy(activities_list[activities_list_names.index(i)][1])
        nodes_list=list(temp_graph1.nodes)

        for k in nodes_list:
            if temp_graph1.nodes[k]['type']!='claim' and temp_graph1.nodes[k]['type']!='release':

                pred_node=list(temp_graph1.predecessors(k))
                succ_node=list(temp_graph1.successors(k))
                temp_graph1.remove_node(k)
                elist=[]
                for pred in pred_node:
                    for succ in succ_node:
                        elist.append((pred,succ))

                temp_graph1.add_edges_from(elist)

```

```
        alternate_dag_unimp.append(copy.deepcopy(temp_graph1))

    size_of_seq=2 #max size of sequence

    #Start: Initial List of sequences to visited

    arr_temp=[]

    new_arr_names_only=[]
    for i in range(size_of_seq):
        arr_temp=list(p for p in product(activities_list_names, repeat=i+1))
        new_arr_names_only=new_arr_names_only+arr_temp

    arr_names=[]

    for i in new_arr_names_only:
        instance_list=[]
        list_to_be_passed=[]
        list_to_be_passed_names=[]
        for j in i:
            instance_list.append(j)
            list_to_be_passed_names.append(j+"_"+str(instance_list.count(j)))

        arr_names.append(list_to_be_passed_names)

    # End: Initial List of sequences to visited

    #Start: Separating important and non-important activities

    unimp_act=[]
    for i in activities_list:
        for j in range(size_of_seq):
            if (i[0]+"_"+str(j+1)) not in imp_act:
                unimp_act.append(i[0]+"_"+str(j+1))


    #End: Separating important and non-important activities
    indexes_to_be_visited=list(range(len(arr_names)))
    len_cntr=[]
    for i in range(1,size_of_seq+1):
        len_cntr.append(pow(len(activities_list_names),i))


    finalReq="requirement req:\n"

    seq_accepted_names=[]

```

```
    no_of_synths=0
    synthPath=Path('se-software-cmdline-win-win-x64-r9682/bin/cif3datasynth.bat')

    start=time.time()
    cntr=1
    for i in arr_names:
        list_to_be_passed_names=i

        if arr_names.index(i) in indexes_to_be_visited:

            indexes_to_be_visited.remove(arr_names.index(i))
            for r in len_cntr:
                if arr_names.index(i)<r:
                    reg=r
                    break

            list_to_be_passed=[]

#           for j in range(len(i)):
#               if i[j] in imp_action:
#                   list_to_be_passed.append([i[j],alternate_dag_imp[activities_list_names.index(new_arr_names_only[arr_names.index(i)][j])]])
#               else:
#                   list_to_be_passed.append([i[j],alternate_dag_unimp[activities_list_names.index(new_arr_names_only[arr_names.index(i)][j])]])

            for j in range(len(i)):
                list_to_be_passed.append([i[j],activities_list[activities_list_names.index(new_arr_names_only[arr_names.index(i)][j])][1]])

            Y=ConvertToAutomata(list_to_be_passed)
            X=Y.stringtowrite8(imp_action)
            fileDirPlant=Path("se-software-cmdline-win-win-x64-r9682/bin/temp_new_3.cif")
            Z=""

            Z_trav=Z_main.splitlines()
            for line in Z_trav:
                if "edge " in line:
                    wrd_arr=line.split()
                    act_to_remove=wrd_arr[wrd_arr.index("edge")+1].split(".")[0]
                    action_to_remove=wrd_arr[wrd_arr.index("edge")+1].split(".")[1]
                    act_available=i
```

```
                if act_to_remove in act_available:
                    if action_to_remove in imp_action[act_to_remove]:
                        temp_line=line.replace('.','_')
                        Z=Z+temp_line+"\n"
                else:
                    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]:
                        if "location " in Z_trav[Z_trav.index(line)-1]:
                            pos=Z.rfind(':')
                            Z=Z[:pos]+";"
            else:
                Z=Z+line+"\n"


        X=X+"\n"+Z
        f=open(fileDirPlant,"w")
        f.write(X)
        f.close()
        # Supervisory synthesis process
        synth = subprocess.run(
                            [synthPath.absolute().as_posix(), fileDirPlant.absolute().as_posix()],
                            capture_output=True,
                            text=True
                            )
        no_of_synths+=1
        print("Sequence visited:"+str(no_of_synths))
        if "ERROR" not in synth.stderr:
            seq_accepted_names.append((list_to_be_passed_names, cntr,unimp_act))
            cntr+=1
            indexes_temp=list(indexes_to_be_visited)
            if len(indexes_to_be_visited)==0:
                break
            for k in indexes_temp:
                if k>=reg:
                    if subseqchecker(arr_names[k], list_to_be_passed_names):
                        if set(diff(arr_names[k], list_to_be_passed_names)).issubset(set(unimp_act)):
                            indexes_to_be_visited.remove(k)
                            if not subseqcheckerend(list_to_be_passed_names, arr_names
```

```
[k]):
                                    seq_accepted_names.append((arr_names[k],cntr,unimp_act))
                                    cntr+=1

                # print(seq_accepted_names)

            else:

                indexes_temp=list(indexes_to_be_visited)
                if len(indexes_to_be_visited)==0:
                    break
                for k in indexes_temp:
                    if k>=reg:
                        if subseqcheckerend(list_to_be_passed_names,arr_names[k]):
                            if set(diff(arr_names[k],list_to_be_passed_names)).issubset(set(unimp_act)):
                                indexes_to_be_visited.remove(k)


    end=time.time()

    print("Total synths: "+str(no_of_synths))
    print("Total time: "+str(end-start))

    pool=Pool(processes=5)
    temp_str_arr=list(pool.map(writetofile,seq_accepted_names))

    finalReq=finalReq+"\tlocation L0:\n\t\tinitial;\n"

    arr1=[element[0] for element in temp_str_arr]
    arr2=[element[1] for element in temp_str_arr]
    finalReq=finalReq+''.join(arr1)
    finalReq=finalReq+'\n'.join(arr2)
    finalReq=finalReq+"\nend"

    fileDirfinal=Path("se-software-cmdline-win-win-x64-r9682/bin/req_final_new.cif")
    fin=open(fileDirfinal,"w")
    fin.write(finalReq)
    fin.close()
```

I Python code: Method V

```python


import networkx as nx
import itertools
import os.path
import subprocess
import copy
import time
from subseqchecker import subseqchecker
from subseqchecker import diff
from ConvertToAutomata import ConvertToAutomata
from activitySequenceExtractor import activitySequenceExtractor
#Start: Extraction of LSAT variables

fileDirReq=os.path.join("F:/Personal/TU Eindhoven/Post Registration/
    Grad Project/LSAT/se-software-cmdline-win-win-x64-r9682/bin/","req
    .cif")
req=open(fileDirReq,"r")
Z_main=req.read()
req.close()

Z_trav=Z_main.splitlines()
imp_act=[]
imp_action={}
for line in Z_trav:
    if "edge " in line:
        wrd_arr=line.split()
        imp_act.append(wrd_arr[wrd_arr.index("edge")+1].split(".")
            [0])
        if wrd_arr[wrd_arr.index("edge")+1].split(".")[0] in
            imp_action:
            imp_action[wrd_arr[wrd_arr.index("edge")+1].split(".")
                [0]].append( wrd_arr[wrd_arr.index("edge")+1].split(".
                ")[1])
        else:
            imp_action[wrd_arr[wrd_arr.index("edge")+1].split(".")
                [0]]=[]
            imp_action[wrd_arr[wrd_arr.index("edge")+1].split(".")
                [0]].append( wrd_arr[wrd_arr.index("edge")+1].split(".
                ")[1])



words_list_master=[]
comment_var=False
with open('Example_Convert_2.activity','r') as file:

```

```python
    # reading each line
    for line in file:

        # reading each word
        for word in line.split():

            #Remove comments
            if word.startswith("//"):
                break

            if word.startswith("/*"):
                comment_var=True

            if "*/" in word:
                comment_var=False
                to_remove=word.split("*/")
                new_word=to_remove[1]
                if new_word:
                    words_list_master.append(new_word)
                continue

            # storing the words
            if comment_var==False:
                if ":" in word:
                    to_remove=word.split(":")
                    new_word=to_remove[0]
                    if new_word:
                        words_list_master.append(new_word)
                    words_list_master.append(":")
                    new_word=to_remove[1]
                    if new_word:
                        words_list_master.append(new_word)
                    continue

                if "->" in word:
                    to_remove=word.split("->")
                    new_word=to_remove[0]
                    if new_word:
                        words_list_master.append(new_word)
                    words_list_master.append("->")
                    new_word=to_remove[1]
                    if new_word:
                        words_list_master.append(new_word)
                    continue


                words_list_master.append(word)

main_Stack=[]
```

```
activities_list=[]
for word in words_list_master:
    main_Stack.append(word)

    if word=="}":
        data_Stack=[]
        main_Stack.pop()
        while(True):
            x=main_Stack.pop()
            if x=="{":
                break
            data_Stack.append(x)

        # print(data_Stack)

        if main_Stack[-1]=="actions":
            if data_Stack:
                temp_graph=nx.DiGraph()
                rev_data_Stack=data_Stack[::-1]

                for word2 in range(len(rev_data_Stack)):
                    if rev_data_Stack[word2]==":":
                        node_name=rev_data_Stack[word2-1]

                        if rev_data_Stack[word2+1]=="claim":
                            type_var="claim"
                            resource_var=rev_data_Stack[word2+2]
                        elif rev_data_Stack[word2+1]=="release":
                            type_var="release"
                            resource_var=rev_data_Stack[word2+2]
                        elif rev_data_Stack[word2+1]=="move":
                            type_var="action"
                            resource_var=rev_data_Stack[word2+2].split(".")[0]
                        else:
                            type_var="action"
                            resource_var=rev_data_Stack[word2+1].split(".")[0]

                        temp_graph.add_node(node_name,resource=resource_var,type=type_var)
                empty_act=False
            else:
                empty_act=True
            # print(temp_graph.nodes)

        if main_Stack[-1]=="flow":
            if data_Stack:
                rev_data_Stack=data_Stack[::-1]
```

```
                for word2 in range(len(rev_data_Stack)-1):
                    if rev_data_Stack[word2]=="->":
                        if rev_data_Stack[word2+1].startswith("|"):
                            sync_nodes=[]
                            for word3 in range(len(rev_data_Stack)):
                                if rev_data_Stack[word3]==rev_data_Stack[word2+1]:
                                    if word3<len(rev_data_Stack)-1:
                                        if rev_data_Stack[word3+1]=="->":
                                            sync_nodes.append(rev_data_Stack[word3+2])
                            for sn in sync_nodes:
                                temp_graph.add_edge(rev_data_Stack[word2-1],sn)


                        else:
                            if not rev_data_Stack[word2-1].startswith("|"):
                                temp_graph.add_edge(rev_data_Stack[word2-1],rev_data_Stack[word2+1])

                empty_act=False
            else:
                empty_act=True

        if main_Stack[-2]=="activity" and not empty_act:
            activities_list.append([copy.deepcopy(main_Stack[-1]),copy.deepcopy(temp_graph)])

activities_list_names=[element[0] for element in activities_list]

#End: Extraction of LSAT variables

size_of_seq=3 #max size of sequence



#Start timer
start=time.time()

activity_path='Example.ctrlsys_statespace.cif'
activityG=activitySequenceExtractor(activity_path)

extracted_seq_list_names=[]
for i in activityG.nodes:
```

```
    # print(i)
    # print(activityG.nodes[i])

    if activityG.nodes[i]['initial']:
        for j in activityG.nodes:
            if activityG.nodes[j]['marked']:

                paths=list(nx.all_simple_edge_paths(activityG, i, j, cutoff=size_of_seq))

                for l in paths:
                    seq_temp=[]
                    for k in l:

                        seq_temp.append(activityG[k[0]][k[1]][k[2]]['name'])

                    extracted_seq_list_names.append(copy.deepcopy(seq_temp))



#Start: Initial List of sequences to visited

arr_temp=[]
new_arr=[]
for i in range(size_of_seq):
    arr_temp=list(p for p in itertools.product(activities_list, repeat=i+1))
    for j in arr_temp:
        new_arr.append(j)

arr_names=[]
for i in new_arr:
    instance_list=[]
    list_to_be_passed=[]
    list_to_be_passed_names=[]
    for j in i:
        instance_list.append(j[0])
        list_to_be_passed_names.append(j[0]+"_"+str(instance_list.count(j[0])))
    arr_names.append(list_to_be_passed_names)

arr=[]
for i in arr_names:
    temp_list=[]
    for j in range(len(i)):

        temp_graph1=copy.deepcopy(new_arr[arr_names.index(i)][j][1])
```

```
        if i[j] not in imp_act:
            nodes_list=list(temp_graph1.nodes)

            for k in nodes_list:
                if temp_graph1.nodes[k]['type']!='claim' and temp_graph1.nodes[k]['type']!='release':
                    remove_node_pred=True
                    remove_node_succ=True

                    for n in temp_graph1.predecessors(k):
                        if temp_graph1.nodes[n]['resource']!=temp_graph1.nodes[k]['resource']:
                            remove_node_pred=False
                            break

                    for n in temp_graph1.successors(k):
                        if temp_graph1.nodes[n]['resource']!=temp_graph1.nodes[k]['resource']:
                            remove_node_succ=False
                            break

                    if remove_node_pred and remove_node_succ:
                        pred_node=temp_graph1.predecessors(k)
                        succ_node=temp_graph1.successors(k)
                        temp_graph1.remove_node(k)
                        elist=[]
                        for pred in pred_node:
                            for succ in succ_node:
                                elist.append((pred,succ))

                        temp_graph1.add_edges_from(elist)

            temp_list.append([i[j],copy.deepcopy(temp_graph1)])

        else:
            nodes_list=list(temp_graph1.nodes)

            for k in nodes_list:
                if temp_graph1.nodes[k]['type']!='claim' and temp_graph1.nodes[k]['type']!='release':
                    if k not in imp_action[i[j]]:

                        remove_node_pred=True
                        remove_node_succ=True

                        for n in temp_graph1.predecessors(k):
                            if temp_graph1.nodes[n]['resource']!=temp_graph1[k]['resource']:
                                remove_node_pred=False
```

```
                                break

                        for n in temp_graph1.successors(k):
                            if temp_graph1.nodes[n]['resource']!=temp_graph1[k]['resource']:
                                remove_node_succ=False
                                break

                        if remove_node_pred and remove_node_succ:
                            pred_node=temp_graph1.predecessors(k)
                            succ_node=temp_graph1.successors(k)
                            temp_graph1.remove_node(k)
                            elist=[]
                            for pred in pred_node:
                                for succ in succ_node:
                                    elist.append((pred,succ))

                            temp_graph1.add_edges_from(elist)

            temp_list.append([i[j],copy.deepcopy(temp_graph1)])



    arr.append(copy.deepcopy(temp_list))

print(len(arr))
#Taking intersection of sequences
arr_names_temp=copy.deepcopy(arr_names)
arr_temp=copy.deepcopy(arr)
arr_names=[]
arr=[]
for i in range(len(arr_names_temp)):
    if arr_names_temp[i] in extracted_seq_list_names:
        arr_names.append(arr_names_temp[i])
        arr.append(arr_temp[i])

print(len(arr))
#  End: Initial List of sequences to visited


#Start: Separating important and non-important activities

unimp_act=[]
for i in activities_list:
    for j in range(size_of_seq):
        if (i[0]+"_"+str(j+1)) not in imp_act:
            unimp_act.append(i[0]+"_"+str(j+1))
```

```python
#End: Separating important and non-important activities


finalReq="requirement req:\n"
cntr=0
seq_accepted=[]
seq_accepted_names=[]
redundant_seq_names=[]
no_of_synths=0

for i in arr:
    list_to_be_passed=i
    list_to_be_passed_names=arr_names[arr.index(i)]

    if list_to_be_passed_names not in redundant_seq_names and list_to_be_passed_names not in seq_accepted_names:
        Y=ConvertToAutomata(list_to_be_passed)
        X=Y.stringtowrite()
        fileDirPlant=os.path.join("F:/Personal/TU Eindhoven/Post Registration/Grad Project/LSAT/se-software-cmdline-win-win-x64-r9682/bin/","temp_.cif")
        Z=""

        Z_trav=Z_main.splitlines()
        for line in Z_trav:
            if "edge " in line:
                wrd_arr=line.split()
                act_to_remove=wrd_arr[wrd_arr.index("edge")+1].split(".")[0]

                act_available=[item[0] for item in list_to_be_passed]
                if act_to_remove in act_available:
                    Z=Z+line+"\n"
                else:
                    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]:
                        if "location " in Z_trav[Z_trav.index(line)-1]:
                            pos=Z.rfind(':')
                            Z=Z[:pos]+";"
            else:
                Z=Z+line+"\n"


        X=X+"\n"+Z
        f=open(fileDirPlant,"w")
        f.write(X)
```

```
        f.close()
        # Supervisory synthesis process
        synth = subprocess.run(
                            ['F:/Personal/TU Eindhoven/Post
                               Registration/Grad Project/LSAT/se-
                               software-cmdline-win-win-x64-r9682/bin
                               /cif3datasynth.bat', fileDirPlant],
                             capture_output=True,
                             text=True
                             )
        no_of_synths+=1
        print("Sequence visited:"+str(list_to_be_passed_names))
        if "ERROR" not in synth.stderr:
            seq_accepted_names.append(list_to_be_passed_names)
            for k in range(1,size_of_seq-len(list_to_be_passed_names)
                +1):
                x=[element for element in itertools.product(unimp_act
                    ,repeat=k)]
                for l in x:
                    y=list(itertools.chain(list_to_be_passed_names,l)
                        )
                    if y in arr_names:
                        seq_accepted_names.append(y)
                    y=list(itertools.chain(l,list_to_be_passed_names)
                        )
                    if y in arr_names:
                        seq_accepted_names.append(y)

            # print(seq_accepted_names)

            for k in range(arr_names.index(list_to_be_passed_names)
                +1,len(arr_names)):
                if subseqchecker(arr_names[k],
                    list_to_be_passed_names) and arr_names[k] not in
                    seq_accepted_names:
                    if all(item in unimp_act for item in diff(
                        arr_names[k],list_to_be_passed_names)):
                            seq_accepted_names.append(arr_names[k])
                    # print("Potential addition :")
                    # print(arr_names[k])


        else:

            print(synth.stderr+"in: "+str(arr.index(i)))
            for k in range(1,size_of_seq-len(list_to_be_passed_names)
                +1):
                x=[element for element in itertools.product(unimp_act
                    ,repeat=k)]
```

```
                for l in x:
                    y=list(itertools.chain(list_to_be_passed_names,l))
                    redundant_seq_names.append(y)
                    y=list(itertools.chain(l,list_to_be_passed_names))
                    redundant_seq_names.append(y)




finalReq=finalReq+"\tlocation L0:\n\t\tinitial;\n"
cntr=1

for i in [item[0] for item in seq_accepted_names]:
    finalReq=finalReq+"\t\tedge "+i+" goto LSeq"+str(cntr)+"_1;\n"
    cntr+=1

cntr=1
for i in seq_accepted_names:
            for j in i:
                if i.index(j)>0:
                    finalReq=finalReq+"\tlocation LSeq"+str(cntr)+"_"+str(i.index(j))+":\n"
                    finalReq=finalReq+"\t\tedge "+j+" goto LSeq"+str(cntr)+"_"+str(i.index(j)+1)+";\n"

            finalReq=finalReq+"\tlocation LSeq"+str(cntr)+"_"+str(i.index(j)+1)+":\n\t\tmarked;\n\n"
            cntr+=1



finalReq=finalReq+"\nend"
print(finalReq)
fileDirfinal=os.path.join("F:/Personal/TU Eindhoven/Post Registration/Grad Project/LSAT/se-software-cmdline-win-win-x64-r9682/bin/","req_final_new_3.cif")
fin=open(fileDirfinal,"w")
fin.write(finalReq)
fin.close()
print("Total synths:"+str(no_of_synths))

#End timer
end=time.time()

print("Time:")
print(end-start)
```

J Auxiliary code: subseqchecker

```python
import networkx as nx
from pathlib import Path


#Larger array comes first
def subseqchecker(arr1,arr2):
    ind=-1
    for i in arr2:
        if i in arr1:
            if arr1.index(i)<=ind:
                return False
            else:
                ind=arr1.index(i)
        else:
            return False

    if ind==-1:
        return False
    else:
        return True

#Smaller array comes first
def subseqcheckerend(arr1,arr2):
    if arr1[0] not in arr2:
        return False
    else:
        part=arr2[arr2.index(arr1[0]):arr2.index(arr1[0])+len(arr1)]
        if part==arr1:
            return True
        else:
            return False



def diff(arr1,arr2):
    return list(set(arr1) - set(arr2)) + list(set(arr2) - set(arr1))


def writetofile(seq_accepted_names):
    finalReq2=""
    finalReq=""
    seq_accepted_name=seq_accepted_names[0]
    cntr=seq_accepted_names[1]
    unimp_act=seq_accepted_names[2]
    for j in seq_accepted_name:
        if seq_accepted_name.index(j)==0:
            finalReq=finalReq+"\t\tedge "+j+" goto LSeq"+str(cntr)+"
```

```
                _1;\n"
        if seq_accepted_name.index(j)>0:
            finalReq2=finalReq2+"\tlocation LSeq"+str(cntr)+"_"+str(seq_accepted_name.index(j))+":\n"
            finalReq2=finalReq2+"\t\tedge "+j+" goto LSeq"+str(cntr)+"_"+str(seq_accepted_name.index(j)+1)+";\n"

    finalReq2=finalReq2+"\tlocation LSeq"+str(cntr)+"_"+str(seq_accepted_name.index(j)+1)+":\n\t\tmarked;\n"
    for k in unimp_act:
        if k not in seq_accepted_name:
            finalReq2=finalReq2+"\t\tedge "+k+";\n"

    return [finalReq,finalReq2]


def automataToAutomataDAG(Z_main):
    automataDAG=nx.DiGraph()
    Z_trav=Z_main.splitlines()
    for line in Z_trav:
        if "location " in line:
            node_temp=line.split()[1]
            node_temp=node_temp.split(":")[0]
            if node_temp not in list(automataDAG.nodes):
                automataDAG.add_node(node_temp,initial=False,marked=False)
            else:
                automataDAG.nodes[node_temp]['initial']=False
                automataDAG.nodes[node_temp]['marked']=False

        if "initial;" in line:
            automataDAG.nodes[node_temp]['initial']=True

        if "marked;" in line:
            automataDAG.nodes[node_temp]['marked']=True

        if "edge " in line:
            edges_temp=line.split()
            edge_name=edges_temp[edges_temp.index('edge')+1]
            if 'goto' not in edges_temp:
                dest_node=node_temp
            else:
                dest_node_temp=edges_temp[edges_temp.index('goto')+1]
                dest_node=dest_node_temp.split(";")[0]

            if (node_temp,dest_node) not in automataDAG.edges:
                automataDAG.add_edge(node_temp,dest_node,edgeName=[edge_name])
            else:
```

```
                automataDAG.edges[node_temp,dest_node]['edgeName'].
                    append(edge_name)

    initial_nodes=[]
    marked_nodes=[]
    for node_ in list(automataDAG.nodes):
        if automataDAG.nodes[node_]['initial']:
            initial_nodes.append(node_)

        if automataDAG.nodes[node_]['marked']:
                marked_nodes.append(node_)

    all_paths=[]
    for source in initial_nodes:
        for dest in marked_nodes:
            if source!=dest:
                paths= list(nx.all_simple_edge_paths(automataDAG,
                    source, dest))
                for path in paths:
                    print(path)
    return automataDAG
```

K Auxiliary code: ConvertToAutomata

```

class ConvertToAutomata:
    #Initialize class instances
    def __init__(self,activities_dict):
        self.activities_dict=activities_dict


    def checkcommon(self,list1,list2):
        for x in list1:
            for y in list2:
                if x==y:
                    return True

        return False

    #function to create string to be written to file
    def stringtowrite(self):
        string_x=""

        for i in self.activities_dict:

            #Writing activity names to string
            string_x=string_x+"\n"


            string_x=string_x+"plant "+i[0]+":"

            #Writing edge names to string
            string_x=string_x+"\n"
            for j in i[1].nodes:
                string_x=string_x+"\tuncontrollable "+j+";\n"

            #Writing state transitions to string
            #dictionary that holds the locations and edges
            string_x=string_x+"\n"
            automatalocs={}

            #list that holds locations to visit
            listoflocs=[]
            listoflocs.append(list(i[1].nodes))

            for j in listoflocs:

                locname=''.join(map(str, j))
                automatalocs[locname]=[]

                for k in j:
```

```
                    if i[1].in_degree(k)==0 or not self.checkcommon(i[1].predecessors(k),j):

                        temp1=j[:]
                        temp1.remove(k)

                        if temp1:
                            temp1_str=' '.join(map(str, temp1))
                        else:
                            temp1_str="_empty_"
                        temp2=[k,temp1_str]
                        automatalocs[locname].append(temp2)
                        if temp1:
                            if temp1 not in listoflocs:
                                listoflocs.append(temp1)



            locstonums=list(automatalocs)
            for i in locstonums:
                string_x=string_x+"\tlocation L"+str(locstonums.index(i))+":\n"
                if locstonums.index(i)==0:
                        string_x=string_x+"\t\tinitial;\n"

                for j in automatalocs[i]:


                    if j[1] not in locstonums:
                        string_x=string_x+"\t\tedge "+j[0]+" goto _empty_"+";\n"
                    else:
                        string_x=string_x+"\t\tedge "+j[0]+" goto L"+str(locstonums.index(j[1]))+";\n"

            string_x=string_x+"\tlocation _empty_:\n\t\tmarked;\nend\n"

        #writing claim release automata
        automatalocs_claim={}
        automatalocs_rel={}
        temp=[]
        for i in self.activities_dict:
            temp.append(i[0])
            for j in i[1].nodes:
                if i[1].nodes[j]['type']=='claim':
                    if i[1].nodes[j]['resource'] in automatalocs_claim:
```

```
                        automatalocs_claim[i[1].nodes[j]['resource']].append([i[0],j])
                    else:
                        automatalocs_claim[i[1].nodes[j]['resource']]=[[i[0],j]]

                if i[1].nodes[j]['type']=='release':
                    if i[1].nodes[j]['resource'] in automatalocs_rel:
                        automatalocs_rel[i[1].nodes[j]['resource']].append([i[0],j])
                    else:
                        automatalocs_rel[i[1].nodes[j]['resource']]=[[i[0],j]]

        for i in automatalocs_claim:
            string_x=string_x+"\n"
            string_x=string_x+"plant availability_"+i+":\n"
            string_x=string_x+"\tlocation unclaimed:\n\t\tinitial;\n\t\tmarked;\n"
            temp3=automatalocs_claim[i][:]
            res = [i for n, i in enumerate(temp3) if i not in temp3[:n]]
            for j in res:
                string_x=string_x+"\t\tedge "+j[0]+"."+j[1]+" goto claimed;\n"

            string_x=string_x+"\tlocation claimed:\n"
            temp3=automatalocs_rel[i][:]
            res = [i for n, i in enumerate(temp3) if i not in temp3[:n]]
            for j in res:
                string_x=string_x+"\t\tedge "+j[0]+"."+j[1]+" goto unclaimed;\n"

            string_x=string_x+"end\n"

        for i in automatalocs_claim:
            cntr=0
            string_x=string_x+"\n"
            string_x=string_x+"plant claimingAutomata_"+i+":\n"
            for j in automatalocs_claim[i]:
                if cntr==0:
                    string_x=string_x+"\tlocation l"+str(cntr)+":\n\t\tinitial;\n"
                else:
                    string_x=string_x+"\tlocation l"+str(cntr)+":\n"

                cntr+=1
                string_x=string_x+"\t\tedge "+j[0]+"."+j[1]+" goto l"
```

```
                +str(cntr)+";\n"

            string_x=string_x+"\tlocation l"+str(cntr)+":\n\t\tmarked
                ;\n"
            string_x=string_x+"end\n"
        return string_x
```

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[i].

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

Saikat Chakraborty.
……………………………………………………..……………..

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

[i] 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