

Systems Design Laboratory

Eclipse Supervisory Control Engineering Toolkit (ESCET)

Matteo Zavatteri

¹Department of Mathematics, University of Padova, ITALY

²Department of Computer Science, University of Verona, ITALY

Motivation for Model-Based Systems Engineering



In the development of systems and supervisory controllers:

- the use of (formal) models and methods for controller design allows for the *validation and verification* of controllers *before* they are actually *implemented and integrated* into the system.
- the approach of *early validation and verification* have been shown to lead to *fewer defects and reduced costs*.

As a result, more and more companies have been increasingly adopting the Model-Based Systems Engineering (MBSE) paradigm.

Eclipse ESCET™

The Eclipse ESCET project provides a model-based approach and toolkit for the development of supervisory controllers.

Learn more

The Eclipse Supervisory Control Engineering Toolkit (Eclipse ESCET™) project is an Eclipse Foundation open-source project that provides a toolkit for the development of supervisory controllers in the Model-Based Systems Engineering (MBSE) paradigm.

- The use of (formal) models for controller design allows for the validation and verification of controllers *before* they are actually *implemented* and *integrated* into the system.
- *Early validation* and *verification* have been shown to lead to fewer defects and reduced costs.

Eclipse ESCET™ / Project ▾ [Home](#) [About](#) [Download](#) [Documentation](#) [Development](#) [Contact/Support](#)

Eclipse ESCET™

The Eclipse ESCET project provides a model-based approach and toolkit for the development of supervisory controllers.

[Learn more](#)

- **The toolkit has a strong focus on model-based design, supervisory controller synthesis, and industrial applicability, for example to cyber-physical systems.**
- **The toolkit supports the entire development process of (supervisory) controllers, from modeling, supervisory controller synthesis, simulation-based validation and visualization, and formal verification, to real-time testing and implementation.**



The Eclipse Supervisory Control Engineering Toolkit (ESCET) was developed approximately over a period of approximately two decades (starting from the early 2000s) at the Eindhoven University of Technology (TU/e) in cooperation with many European and national projects.



Eclipse ESCET™ / Project ▾ **Home** About Download Documentation Development Contact/Support

Eclipse ESCET™

The Eclipse ESCET project provides a model-based approach and toolkit for the development of supervisory controllers.

[Learn more](#)

In 2021, Eclipse ESCET became an independent Eclipse Foundation open source project, and is no longer formally associated with the TU/e.

Eclipse ESCET Supervisory Control Engineering Toolkit (ESCET)

The screenshot shows the Eclipse ESCET website homepage. At the top is a yellow navigation bar with the text "Eclipse ESCET™ / Project" followed by a dropdown arrow and links for "Home", "About", "Download", "Documentation", "Development", and "Contact/Support". In the top right corner, it says "Version: v0.4". The main heading is "Eclipse ESCET™" in a large, bold, black font. Below it is a subtitle: "The Eclipse ESCET project provides a model-based approach and toolkit for the development of supervisory controllers." A yellow "Learn more" button is centered below the subtitle. The section "Languages and tools" is highlighted with a light blue background. It contains three items, each with a scissors icon in a colored circle: "CIF" (blue circle), "Chi" (green circle), and "ToolDef" (purple circle). Each item has a short description and a "Learn more" button in a matching color.

Eclipse ESCET™ / Project ▾ [Home](#) [About](#) [Download](#) [Documentation](#) [Development](#) [Contact/Support](#)

Version: v0.4

Eclipse ESCET™

The Eclipse ESCET project provides a model-based approach and toolkit for the development of supervisory controllers.

[Learn more](#)

Languages and tools

-  **CIF**
CIF is a modeling language and extensive toolset supporting the entire development process of supervisory controllers.
[Learn more](#)
-  **Chi**
Chi is a modeling language and toolset to analyze the performance of supervisory controllers.
[Learn more](#)
-  **ToolDef**
ToolDef is a cross-platform and machine-independent scripting language to automate CIF and Chi tools.
[Learn more](#)

Eclipse ESCET is based on CIF: the *Compositional Interchange Format* for hybrid systems. CIF is an automata-based modeling language for the specification of discrete event, timed, and hybrid systems.

Eclipse ESCET™ / Project ▾ **Home** About Download Documentation Development Contact/Support

Eclipse ESCET™

The Eclipse ESCET project provides a model-based approach and toolkit for the development of supervisory controllers.

[Learn more](#)

- **Modeling of hybrid systems**
- **Graphical user interface**
- **Simulation**
- **Finite state automata operations**
- **Controller synthesis for (extended) finite state automata**
- **PLC code generation**
- **Employed in many real-world case studies**

4TC00 Home

Search docs

COURSE INFO

- Study guide
- Course schedule
- Downloads
- FAQ

MODULES

- 1: Tooling, Basics, Groups
- 2: Types 1, Data
- 3: Reuse 1, Tims, SVG
- 4: Twin tooling, MBSE
- 5: Channels, Types Reuse 2
- 6: Functions, Stochastics
- 7: Git basics, Git teams
- 8: Git branches, Git advanced

EXAMINATIONS

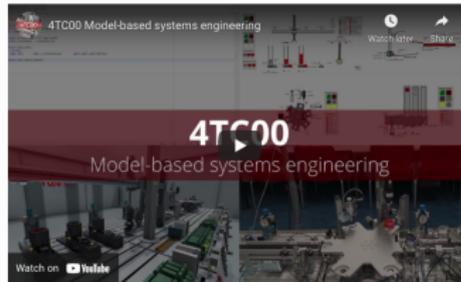
- Intermediate exam
- Midterm assignment
- Final assignment
- Peer review
- Fraud

FESTO WORKSTATIONS

- Festo hardware manual
- Festo lab and videos
- Digital twins
- TwinCAT setup
- TwinCAT control of digital twin
- TwinCAT control of Festo WS
- TwinCAT tips and tricks

4TC00 Model-based systems engineering

4TC00 Model-based systems engineering



4TC00 Model-based systems engineering is a third year bachelor course given by the [Control Systems Technology Group](#)¹² of the Mechanical Engineering department at the [Eindhoven University of Technology](#)¹². The course treats the development process of supervisory controllers, from specification, simulation-based validation and specification, to code generation and real-time testing on [3D digital twins](#) and on real Festo workstations. The video at the top of the page gives a short, 2.5 minute overview of the course.

This website is freely accessible to anyone interested in model-based systems engineering. A small part of the course, including the quizzes and group creation, is available only via [Canvas](#)¹². This part is accessible only to students from the Eindhoven University of Technology who are officially taking this course.

To ensure that you always see the latest version of each web page, it may be necessary to [refresh the page](#)¹², or to [clear the cache](#)¹² (which refreshes all your webpages). When in doubt, you can find the date of each page at the bottom.

[STUDY GUIDE](#) →¹²

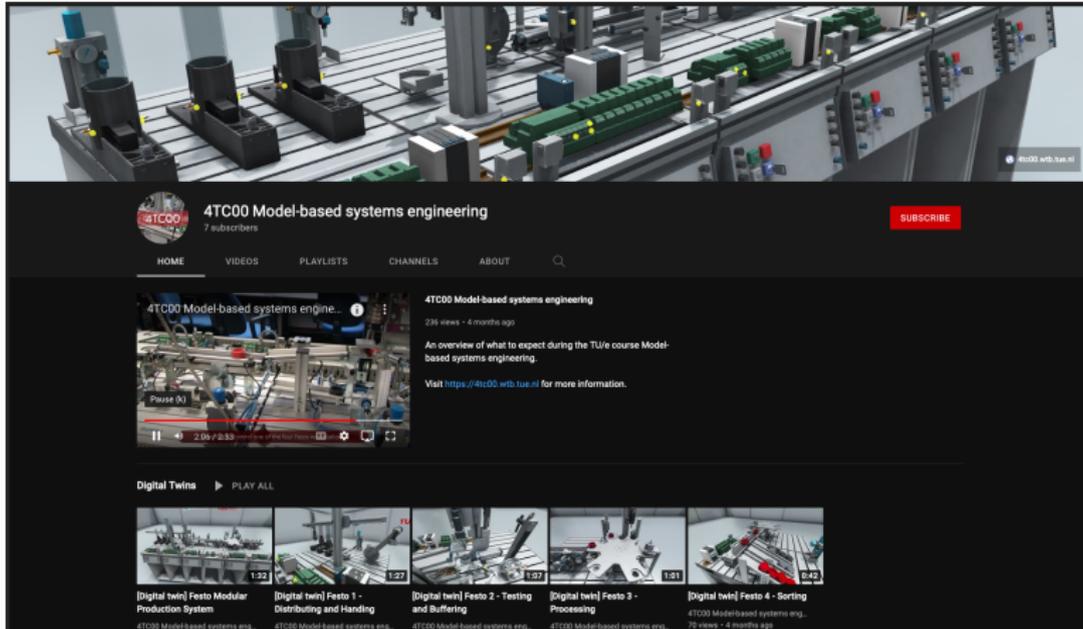
[SCHEDULE](#) →¹²

[MODULES](#) →¹²

Employed in the course 4TC00 Model-Based Systems Engineering (bachelor degree, 3rd year) Eindhoven University of Technology (TU/e).

<https://cstweb.wtb.tue.nl/4tc00/index.html>

Eindhoven University of Technology (TU/e) - 4TC00 course



The image shows a screenshot of a YouTube channel page. At the top, there is a banner image of a 3D CAD model of a modular production system. Below the banner, the channel name '4TC00 Model-based systems engineering' is displayed with a profile picture and '7 subscribers'. A red 'SUBSCRIBE' button is on the right. Navigation tabs for 'HOME', 'VIDEOS', 'PLAYLISTS', 'CHANNELS', and 'ABOUT' are visible. The main content area features a video player with a thumbnail of a factory floor. To the right of the player, the video title '4TC00 Model-based systems engineering' is shown, along with '239 views · 4 months ago'. The description reads: 'An overview of what to expect during the TU/e course Model-based systems engineering. Visit <https://4tc00.web.tue.nl> for more information.' Below the video player, there is a 'Digital Twins' playlist section with a 'PLAY ALL' button and five video thumbnails. The thumbnails are titled: 'Digital twin] Festo Modular Production System', 'Digital twin] Festo 1 - Distributing and Handling', 'Digital twin] Festo 2 - Testing and Buffering', 'Digital twin] Festo 3 - Processing', and 'Digital twin] Festo 4 - Sorting'. Each thumbnail includes a duration and a view count.

Check out the youtube channel for videos, examples, and more.

https://www.youtube.com/channel/UC1lkrIkRkgtbYDul9BwI_Bw

Compositional Interchange Format (CIF)

The screenshot shows the Eclipse ESCET / CIF website. At the top is a blue navigation bar with the text "Eclipse ESCET™ / CIF" and a dropdown arrow, followed by links for "Home", "About", "Download", "Documentation", "Development", and "Contact/Support". On the right side of the page, the text "Version: v0.4" is displayed. The main content area has a light blue background and features a large "CIF" heading. Below this heading, there are two paragraphs of text: "CIF is a declarative modeling language for the specification of discrete event, timed, and hybrid systems as a collection of synchronizing automata." and "The CIF tooling supports the entire development process of controllers, including among others specification, supervisory controller synthesis, simulation-based validation and visualization, verification, real-time testing, and code generation." A blue "Learn more" button is positioned below the second paragraph. The "Features" section follows, with a large "Features" heading. It contains three columns, each with an icon and a title: 1. A hamburger menu icon followed by "Powerful language", with text describing a powerful declarative automata-based modeling language. 2. A scissors icon followed by "Extensive tools", with text describing support for the entire development process of controllers. 3. A lightbulb icon followed by "World-class algorithms", with text describing world-class algorithms for synthesis. A blue "Learn more" button is located at the bottom of the features section.

Eclipse ESCET™ / CIF ▾ Home About Download Documentation Development Contact/Support

Version: v0.4

CIF

CIF is a declarative modeling language for the specification of discrete event, timed, and hybrid systems as a collection of synchronizing automata.

The CIF tooling supports the entire development process of controllers, including among others specification, supervisory controller synthesis, simulation-based validation and visualization, verification, real-time testing, and code generation.

[Learn more](#)

Features

-  **Powerful language**
CIF features a powerful declarative automata-based modeling language for the specification of discrete event, timed and hybrid systems.
-  **Extensive tools**
The CIF tooling supports the entire development process of controllers, from specification to code generation.
-  **World-class algorithms**
The CIF toolset features world-class algorithms for synthesis of supervisory controllers. Focus on the 'what' rather than the 'how'!

[Learn more](#)

<https://www.eclipse.org/escet/cif/>

About CIF

CIF is a rich state machine language with the following main features:

- **Modular specification with synchronized events and communication between automata**
- **Many data types are available (booleans, integers, reals, tuples, lists, arrays, sets, and dictionaries), combined with a powerful expression language for compact variable updates.**
- **Text-based specification of the automata, with many features to simplify modeling large non-trivial industrial systems.**
- **Primitives for supervisory controller synthesis are integrated in the language.**

About CIF

The CIF tooling supports the entire development process of controllers, including among others specification, supervisory controller synthesis, simulation-based validation and visualization, verification, real-time testing, and code generation.

Highlights of the CIF tooling include:

- Text-based editor that allows to easily specify and edit models.
- Feature-rich powerful event-based and data-based supervisory controller synthesis tool.
- A simulator that supports both interactive and automated validation of specifications. Powerful visualization features allow for interactive visualization-based validation.
- Conversion to other formal verification tools such as mCRL2 and UPPAAL.
- Implementation language code generation (PLC languages, Java, C, and Simulink) for real-time testing and implementation of controllers.

About CIF

Supervisory controller synthesis is a key feature of CIF.

- **It involves the automatic generation of supervisory controllers from a specification of the uncontrolled system and the (safety) requirements that the controller needs to enforce.**
- **This moves controller design from “how should the implementation work” to “what should the controller do”.**
- **Implementation of the controller is achieved through code generation, reducing the number of errors introduced at this stage.**

About CIF

- **CIF has been applied in industry, for various domains, including the medical, semiconductor and public works (infrastructure) domains.**
- **The main application area of CIF is the development of supervisory controllers.**
- **The language and tools are generic, and can be used to work with state machines in general for various other purposes.**

About CIF

- **The CIF language and tools are being developed as part of the Eclipse ESCET open-source project.**
- **The CIF tools are part of the Eclipse ESCET toolkit.**

ToolDef: An Integrated Scripting Language

Eclipse ESCET™ / ToolDef ▾ [Home](#) [About](#) [Download](#) [Documentation](#) [Development](#) [Contact/Support](#) Version: v0.4

ToolDef

Tired of scripting with Windows batch files and Linux shell scripts?

ToolDef is a cross-platform scripting language with the simplicity of Python and the power of Java.

[Learn more](#)

Features

 **Intuitive language**

ToolDef features a simple and intuitive Python-inspired syntax that makes it easy to write scripts.

 **Reduce mistakes**

Static typing reduces simple mistakes, compared to Windows batch files, Linux shell scripts and Python.

 **Powerful tools**

ToolDef features many built-in data types and tools, and integrates well with Java and the Eclipse ESCET tools.

[Learn more](#)

Getting started

The ToolDef tooling is part of the Eclipse ESCET toolkit.
It is available for Windows, Linux and macOS, portable and ready to go.

[Download](#)

<https://www.eclipse.org/escet/tooldef/>

About ToolDef

ToolDef allows us to:

- **write scripts using a simple and intuitive syntax, loosely based on the better aspects of Python.**
- **catch simple mistakes early on due to static typing.**
- **work with data of all kinds, using a large number of built-in data types.**
- **manipulate data and paths, work with files and directories, and much more, with over 80 built-in tools.**
- **share your tools as ToolDef libraries.**
- **unleash the full power of Java by importing any Java static method and using it like any other ToolDef tool.**

Eclipse ESCET™ downloads

The Eclipse ESCET toolkit contains the tooling for CIF, Chi and ToolDef.



IDE

The Eclipse ESCET IDE offers the most complete and integrated experience, from convenient editing to execution of the various tools. It suits most users.



Command line

The Eclipse ESCET command line tools allow execution on headless systems and also support integration with other tools, for advanced usage.



P2 update site

The Eclipse ESCET P2 update site contains all the toolkit's plugins and features, for easy integration into OSGI-based applications.

The Eclipse ESCET toolkit includes both the IDE (all platforms) and the command line tools (Windows and Linux only). It is portable, so just download, extract, and run it, to get started quickly.

Version: v0.4 ([release notes](#))

Windows x64 (64-bit) [Download](#) [Mirrors](#)

Linux x64 (64-bit) [Download](#) [Mirrors](#)

macOS x64 (64-bit) [Download](#) [Mirrors](#)

Eclipse ESCET is also available as an Eclipse P2 update site:
<https://download.eclipse.org/escet/v0.4/update-site/>

<https://www.eclipse.org/escet/download.html>

ESCET - Integrated Development Environment

The screenshot displays the Eclipse ESCET IDE interface. The main editor shows the code for `tank.cif`, which defines a tank system with a controller and various outputs. The code includes a group `tank` with a constant `V = 10.0`, a controller `n` that switches between `opened` and `closed` states, and several `svgout` statements for visualizing the tank's state and flow rates.

```
15: group tank:
16:   cont V = 10.0;
17:   alg real Qi = controller.n * 5.0;
18:   alg real Qo = sqrt(V);
19:   equation V' = Qi - Qo;
20:
21:   svgout id "water" attr "height" value 7.5 * V;
22:   svgout id "v" text value fmt("%.1f", V);
23:   svgout id "Qi" text value fmt("Qi = %.1f", Qi);
24:   svgout id "Qo" text value fmt("Qo = %.1f", Qo);
25: end
26:
27: automation controller:
28:   alg int n;
29:
30:   location closed:
31:     initial;
32:     equation n = 0;
33:     edge when tank.V <= 2 goto opened;
34:
35:   location opened:
```

The `Plot Visualizer` window shows a graph of the tank's state over time. The x-axis represents time (0 to 35), and the y-axis represents the tank's state (0 to 10). The plot shows the tank's volume `V` (yellow line) oscillating between 0 and 10, and the flow rates `Qi` (green line) and `Qo` (blue line) oscillating between 0 and 5.0.

The `SVG Visualizer` window shows a schematic diagram of the tank system. A valve `n` is shown above the tank, with an input flow rate `Qi = 5.0` and an output flow rate `Qo = 3.0`. The tank's volume `V` is shown as 9.0.

The `State Visualizer` window shows the current state of the simulation. The table below summarizes the state variables:

Name	Value
time	39.599999999999994
controller	opened
controller.n	1
tank.Qi	5.0

The `Console` window shows the simulation status: `ToolDef interpreter [TERMINATED after 42s 638ms] /CIFExamples-0.1.0.qualifier/hybrid/tank/tank.toolDef (started at 2020-05-07 14:28:53.278)`. The simulation was terminated per the user's request.

317M of 778M

CIF Basics - Automata

- CIF models consist of components
- Each component represents a part of the system
- Components are modeled as automata.
- Automata are the basics of CIF constructs
- The name “locations” comes from hybrid automata
- State = location + values of continuous variables
- For finite state automata, states = locations

```
automaton Component1:  
  location L1:  
    ...  
  
  location Ln:  
    ...  
end
```

...

```
automaton ComponentM:  
  location L1:  
    ...  
  
  location Ln:  
    ...  
end
```



CIF Basics - Locations

Locations can be:

- **initial**
- **marked**
- **initial and marked**
- **none of the previous**

```
automaton Component:  
  location L1:  
    initial;  
  ...  
  
  location L2:  
    ...  
  
  location L3:  
    ...  
  
end
```



Location L1 is initial, whereas locations L2 and L3 are neither initial nor marked.

CIF Basics - Locations

Locations can be:

- **initial**
- **marked**
- **initial and marked**
- **none of the previous**

```
automaton Component:  
  location L1:  
    initial;  
  ...  
  
  location L2:  
    marked;  
  ...  
  
  location L3:  
    ...  
  
end
```



Location L1 is initial, location L2 is marked, whereas location L3 is neither initial nor marked.

CIF Basics - Locations

Locations can be:

- **initial**
- **marked**
- **initial and marked**
- **none of the previous**

```
automaton Component:  
  location L1:  
    initial; marked;  
    ...  
  
  location L2:  
    marked;  
    ...  
  
  location L3:  
    ...  
  
end
```



Location L1 is both initial and marked, location L2 is marked, whereas location L3 is neither initial nor marked.

Events can be:

- local
- global

```
event a;  
automaton Component:  
  event b;  
  event c;  
  location L1:  
    initial; marked;  
    ...  
  
  location L2:  
    ...  
end
```

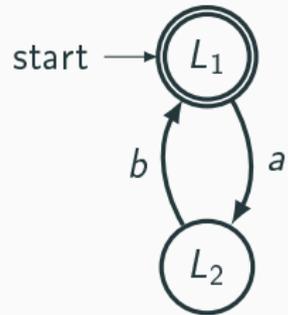
Event a is global, whereas events b and c are local.

CIF - Edges (i.e., Transitions)

Edges:

- model transitions
- have a unique source
- have a unique target
- are associated to events

```
event a;  
automaton Component:  
  event b;  
  location L1:  
    initial; marked;  
    edge a goto L2;  
  location L2:  
    edge b goto L1;  
end
```



We have two transitions

- 1) A transition from L1 to L2 executing event a
- 2) A transition from L2 to L1 executing event b

So basically, the automaton will continue executing

a,b,a,b,a,b,a,b,a,b,a,b,...

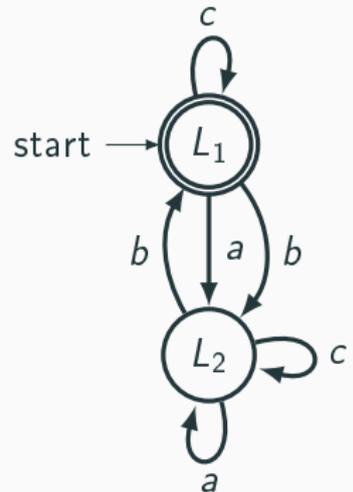
CIF - Edges (cont.)

Edges:

- model transitions
- have a unique source
- have a unique target
- are associated to events

```
event a;
automaton Component:
  event b;
  event c;
  location L1:
    initial; marked;
    edge a goto L2;
    edge b goto L2;
    edge c goto L1;

  location L2:
    edge b goto L1;
    edge a goto L2;
    edge c goto L2;
end
```



We have 6 transitions

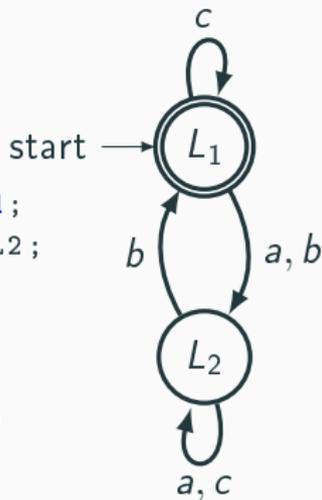
- 1) A transition from L1 to L2 executing event a
- 2) A transition from L1 to L2 executing event b
- 3) A self-loop transition at L1 executing event c
- 4) A transition from L2 to L1 executing event b
- 5) A self-loop transition at L2 executing event a
- 6) A self-loop transition at L2 executing event c

CIF - Edges - Short Notations

```
event a;  
automaton Component:  
  event b;  
  event c;  
  location L1:  
    initial; marked;  
    edge a goto L2;  
    edge b goto L2;  
    edge c goto L1;  
  
  location L2:  
    edge b goto L1;  
    edge a goto L2;  
    edge c goto L2;  
  
end
```

⇒

```
event a;  
automaton Component:  
  event b,c;  
  location L1:  
    initial; marked;  
    edge a,b goto L2;  
    edge c;  
  
  location L2:  
    edge b goto L1;  
    edge a,c;  
  
end
```



General syntax:

```
edge a[,b,...] [goto Lj];
```

CIF Basics - Nameless Locations

When an automaton
has a single location:

- we can omit the name of the location
- only self-loop transitions are allowed (no need to specify the target)

```
automaton Component:  
  event a;  
  location:  
    initial; marked;  
    edge a;  
end
```



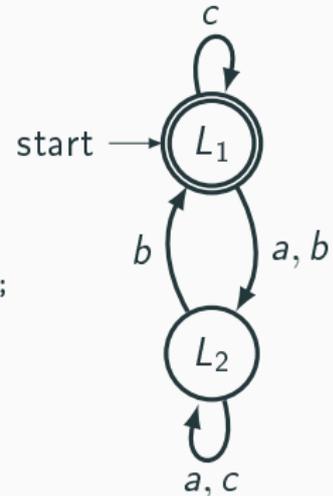
CIF - Implicit Alphabet

Automaton Alphabet:

- Not defined in the code
- The union of all events appearing in edge statements (of that automaton)

```
event a, d;
```

```
automaton Component:  
  event b, c;  
  location L1:  
    initial; marked;  
    edge a, b goto L2;  
    edge c;  
  
  location L2:  
    edge b goto L1;  
    edge a, c;  
  
end
```


$$\Sigma := \{a, b, c\}$$

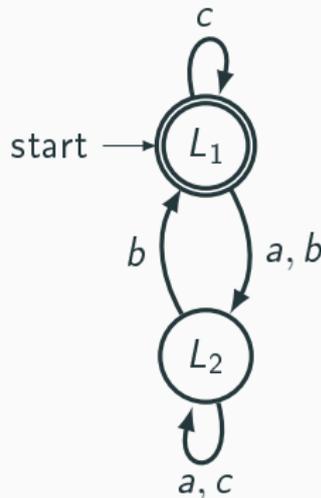
Event d is always executable in a concurrent execution with some other automaton that can execute d.

CIF - Explicit Alphabet

Automaton Alphabet:

- Explicitly defined
- No obligation of using alphabet events on transitions

```
event a, d;  
  
automaton Component:  
  event b, c;  
  alphabet a, b, c, d;  
  location L1:  
    initial; marked;  
    edge a, b goto L2;  
    edge c;  
  
  location L2:  
    edge b goto L1;  
    edge a, c;  
  
end
```



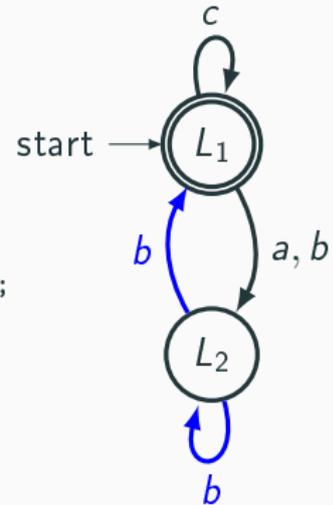
$\Sigma := \{a, b, c, d\}$

Watch out: being in the alphabet of the automaton but not taking part in any of its transitions, event d is never executable in a concurrent execution with some other automaton that can execute d .

CIF - Non Determinism

- More edges for the same event.
- Multiple initial states not supported
- ϵ transitions not supported

```
event a;  
  
automaton Component:  
  event b;  
  location L1:  
    initial; marked;  
    edge a,b goto L2;  
    edge b;  
  
  location L2:  
    edge b goto L1;  
    edge b;  
  
end
```



By executing event b from location L2 we can either remain there or move to L1.

Supervisory Control - Event Controllability

Beside globality or locality, events are also partitioned in:

- controllable (default)
- uncontrollable

```
controllable event a;  
automaton Component:  
    uncontrollable event b;  
    event c;  
    location L1:  
        initial; marked;  
        ...  
  
    location L2:  
        ...  
  
end
```

Event a is global and controllable, b is local and uncontrollable, whereas c is local and controllable.

Supervisory Control - Events - Short Notation

```
controllable event a;
automaton Component:
  event b;
  event c;
  uncontrollable event d;
location L1:
  initial; marked;
  ...

location L2:
  ...

end
```

⇒

```
controllable a;
automaton Component:
  event b, c;
  uncontrollable d;
location L1:
  initial; marked;
  ...

location L2:
  ...

end
```

General syntax

Controllable events

```
event a[,b,...];
controllable event a[,b,...];
controllable a[,b,...];
```

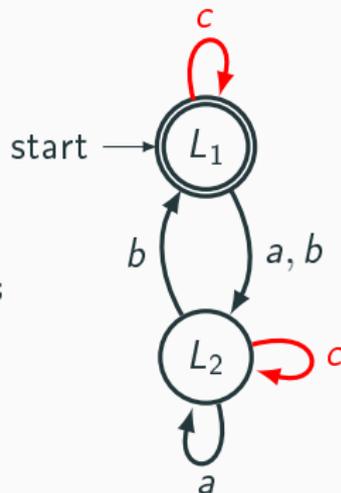
Uncontrollable events

```
uncontrollable event a[,b,...];
uncontrollable a[,b,...];
```

CIF - Edges - Uncontrollable Transitions

Edges related to uncontrollable events model uncontrollable transitions.

```
event a;  
automaton Component:  
  event b;  
  uncontrollable c;  
  location L1:  
    initial; marked;  
    edge a,b goto L2;  
    edge c;  
  
  location L2:  
    edge b goto L1;  
    edge a,c;  
  
end
```



General syntax:

```
edge a[,b,...] [goto Lj];
```

Automata can be of the following types

- Plant
- Requirement
- Supervisor

```
plant automaton C:
```

```
...
```

```
end
```

```
requirement automaton R:
```

```
...
```

```
end
```

```
supervisor automaton S:
```

```
...
```

```
end
```

Supervisory Control - Automata Types - Short Notation

```
plant automaton C:
```

```
...
```

```
end
```

```
requirement automaton R:
```

```
...
```

```
end
```

```
supervisor automaton S:
```

```
...
```

```
end
```

⇒

```
plant C:
```

```
...
```

```
end
```

```
requirement R:
```

```
...
```

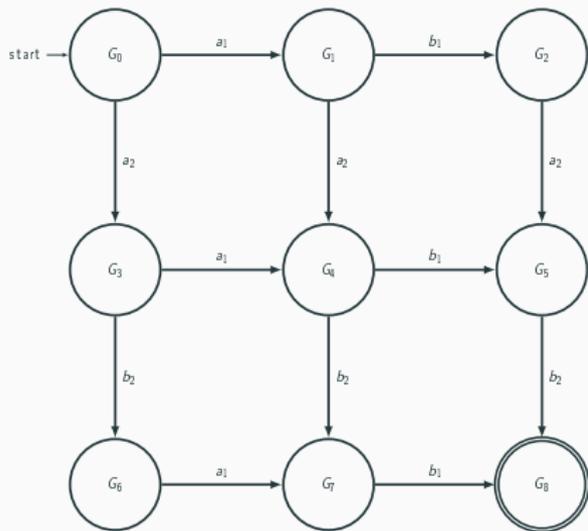
```
end
```

```
supervisor S:
```

```
...
```

```
end
```

The Database Concurrency Example

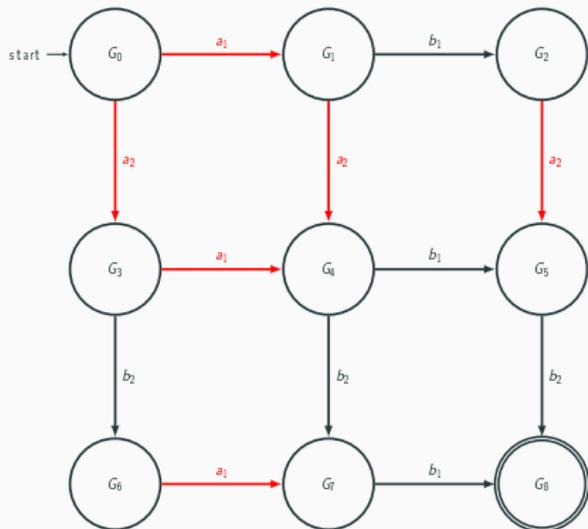


- Two transactions
 - $T_1 := a_1 b_1$
 - $T_2 := a_2 b_2$
- (x_i : some operation by transaction i on record x)
- G_0 is the initial state
- G_8 is the marked state (=completion of T_1 and T_2)

"From the theory of database concurrency control, it can be shown that the only admissible strings are those where a_1 precedes a_2 if and only if b_1 precedes b_2 ."

Cassandras, Lafortune - Introduction to Discrete Event Systems

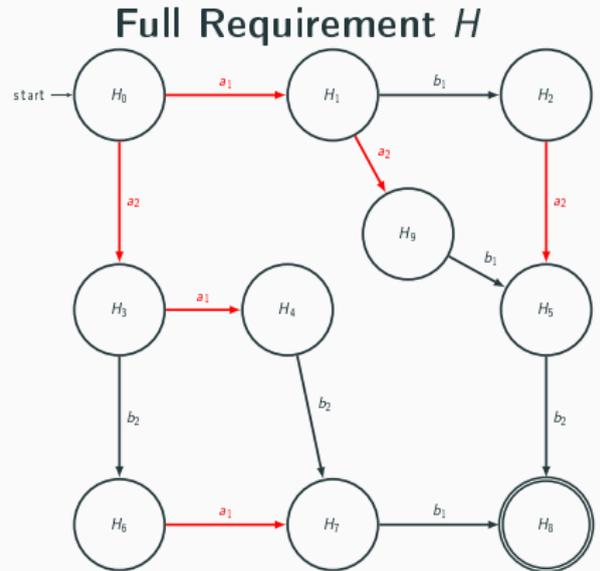
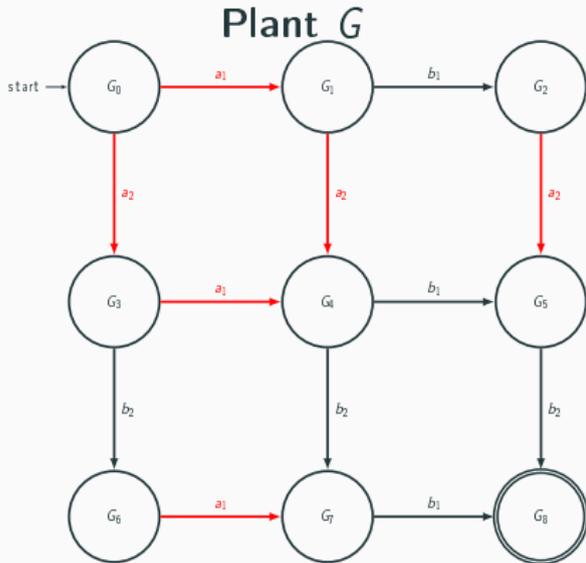
Example 1



- Events a_1, a_2 are uncontrollable
- Events b_1, b_2 are controllable
- G_0 is the initial state
- G_8 is the marked state

Requirement: a_1 precedes a_2 if and only if b_1 precedes b_2

Example 1 - Plant and Requirement

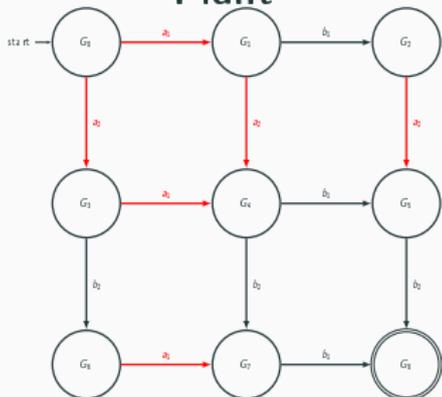


Requirement: a_1 precedes a_2 if and only if b_1 precedes b_2

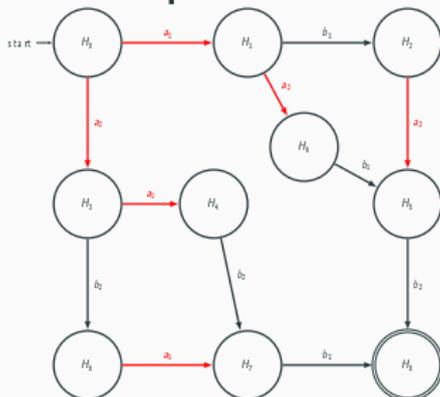
Note: Full Requirement := Plant || Essential Requirement

Example 1 - Controller Synthesis

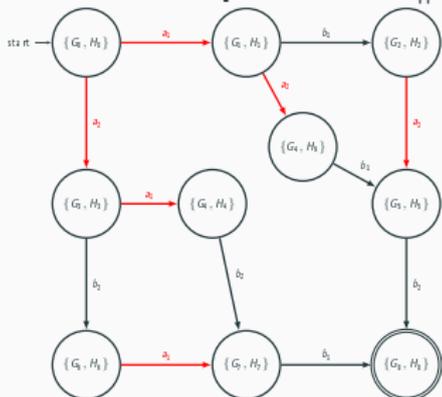
Plant



Requirement

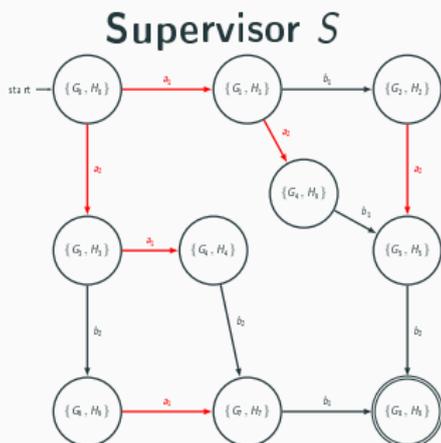
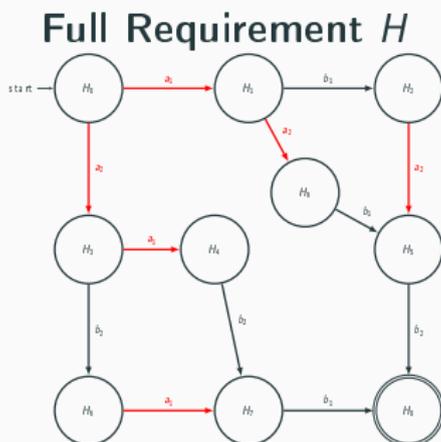
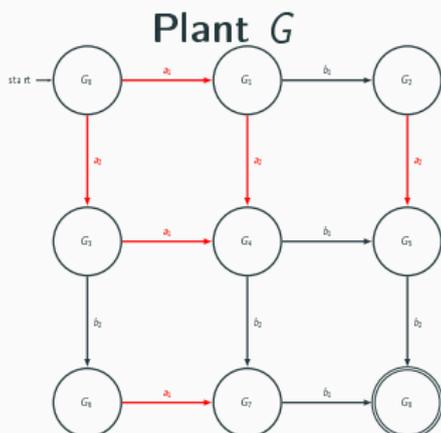


Tentative Supervisor $G \parallel H$



Requirement: a_1 precedes a_2 if and only if b_1 precedes b_2

Example 1 - Controller Synthesis

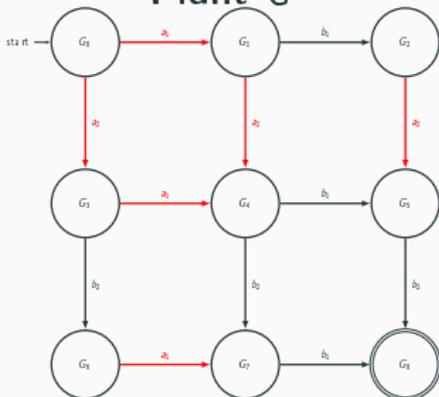


Requirement: a_1 precedes a_2 if and only if b_1 precedes b_2

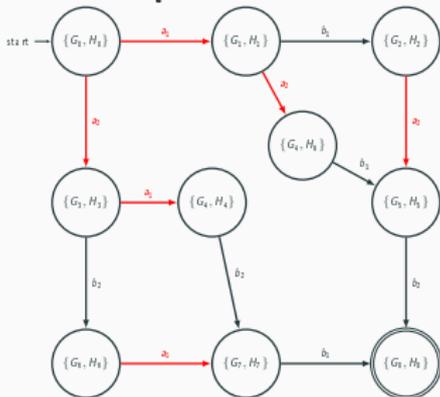
- No states to remove
- ⇓
- Final supervisor

Example 1 - Controller Synthesis

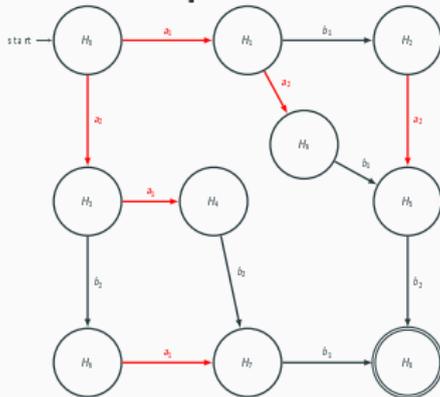
Plant G



Supervisor S



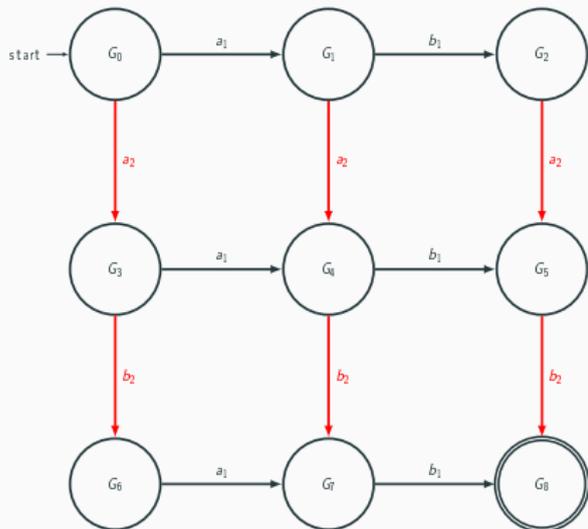
Full Requirement H



Control Policy:

- If the **plant** gets to G_4 by executing $a_1 a_2$, then S disables b_2 .
- If the **plant** gets to G_4 by executing $a_2 a_1$ and S disables b_1 .

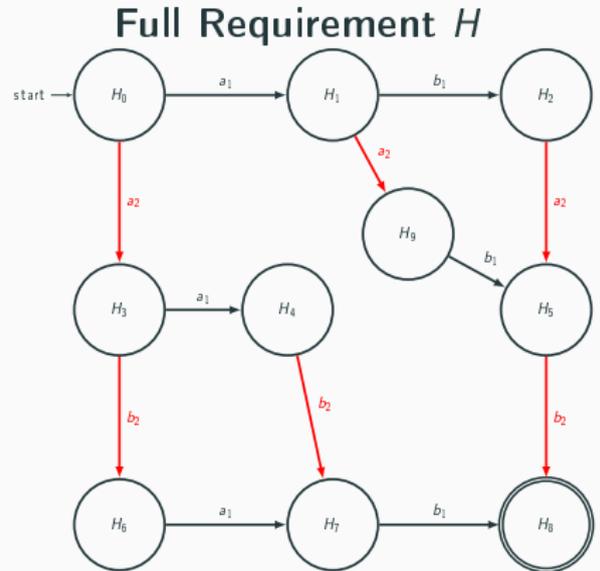
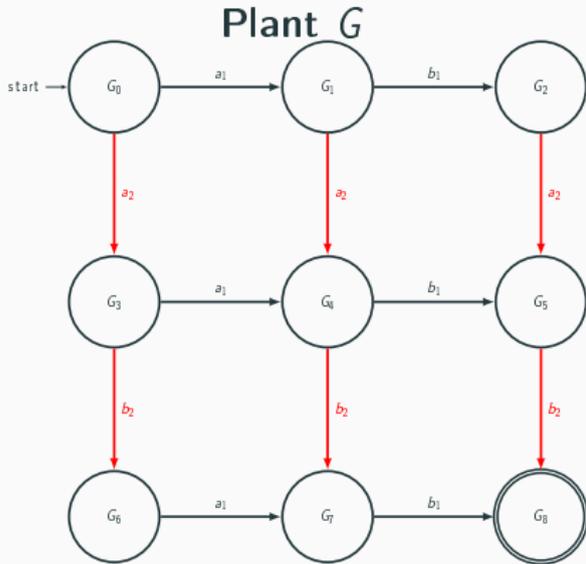
Example 2



- Events a_1, b_1 are controllable
- Events a_2, b_2 are uncontrollable
- G_0 is the initial state
- G_8 is the marked state

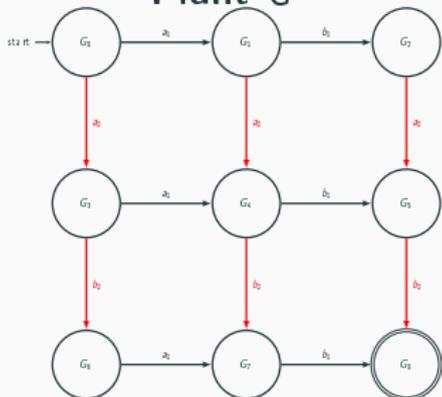
Same Requirement: a_1 precedes a_2 if and only if b_1 precedes b_2

Example 2 - Plant and Requirement

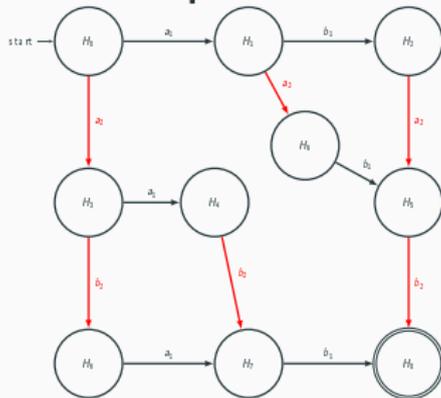


Example 2 - Controller Synthesis

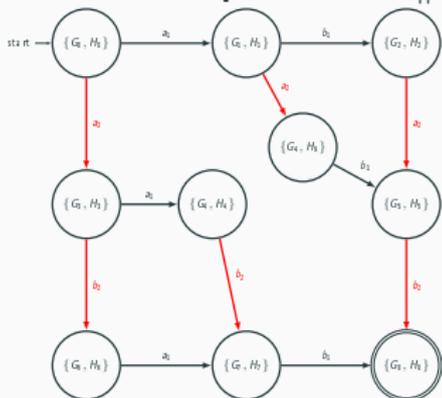
Plant G



Full Requirement H

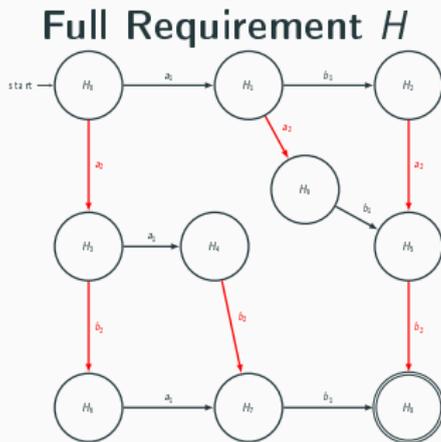
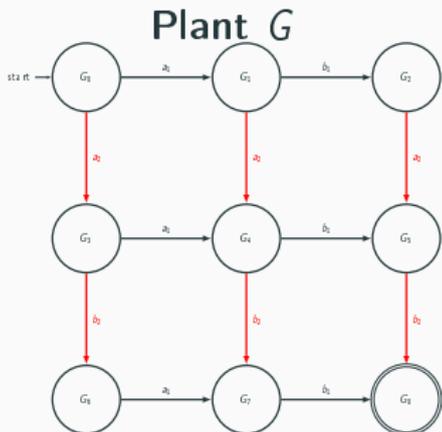


Tentative Supervisor $G \parallel H$

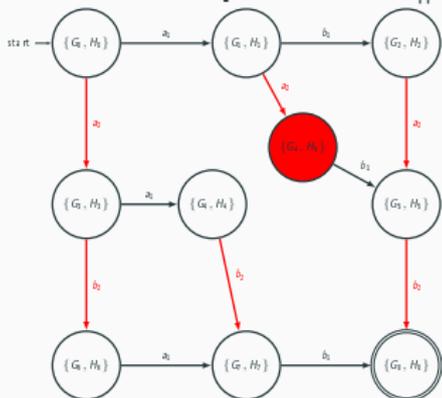


- Any problems?

Example 2 - Controller Synthesis

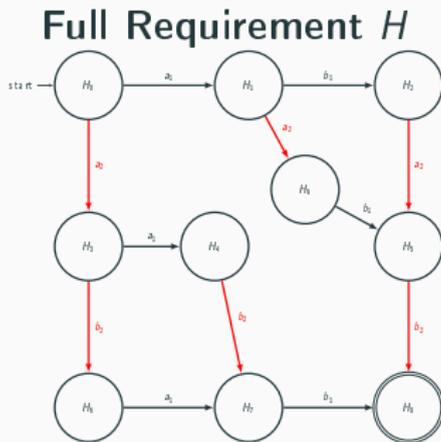
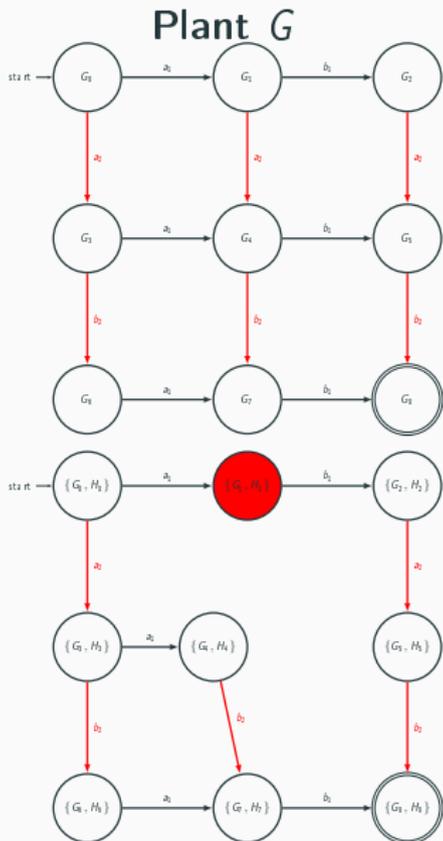


Tentative Supervisor $G \parallel H$



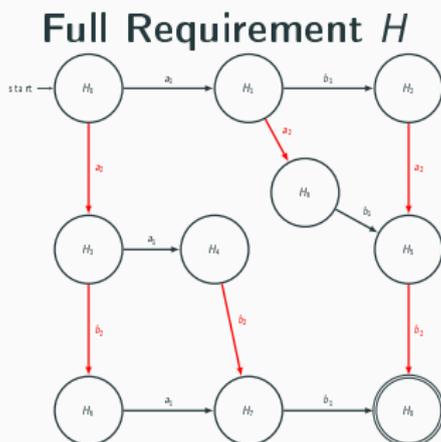
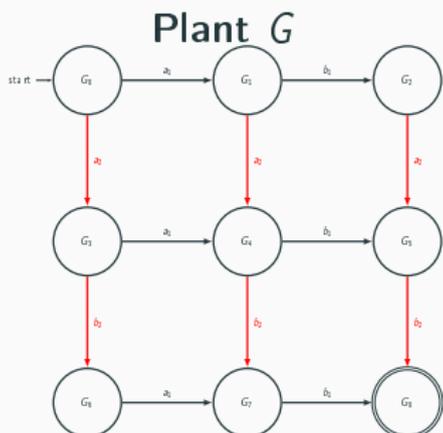
- $\{G_4, H_9\}$ is uncontrollable

Example 2 - Controller Synthesis

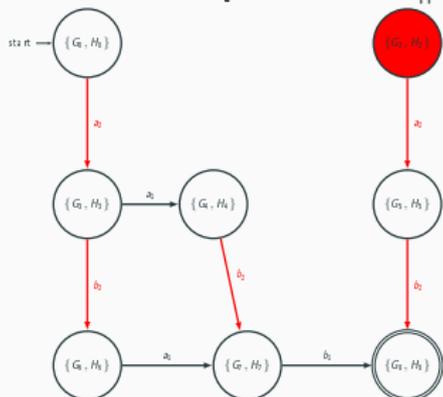


- $\{G_1, H_1\}$ is uncontrollable

Example 2 - Controller Synthesis

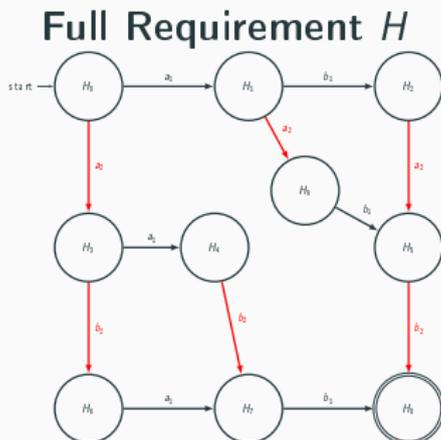
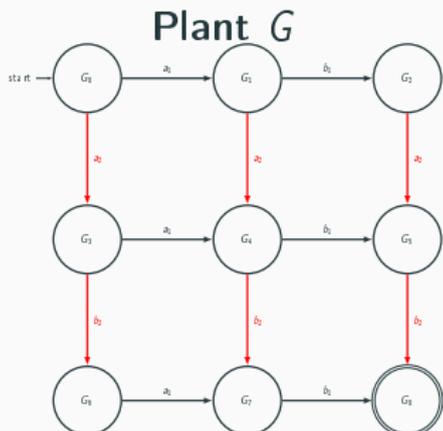


Tentative Supervisor $G \parallel H$

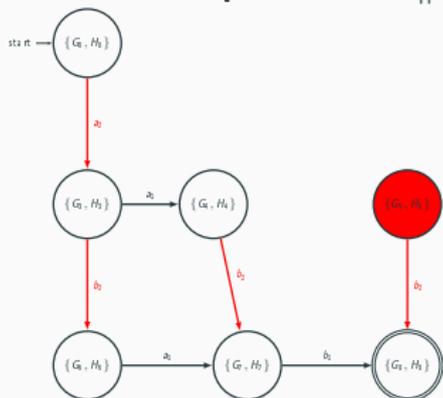


- $\{G_2, H_2\}$ is non-accessible (unreachable from the initial state $\{G_0, H_0\}$)

Example 2 - Controller Synthesis

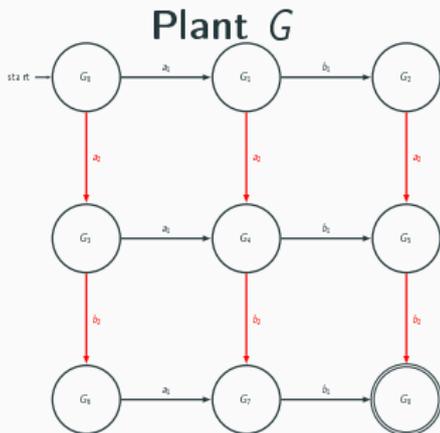


Tentative Supervisor $G \parallel H$

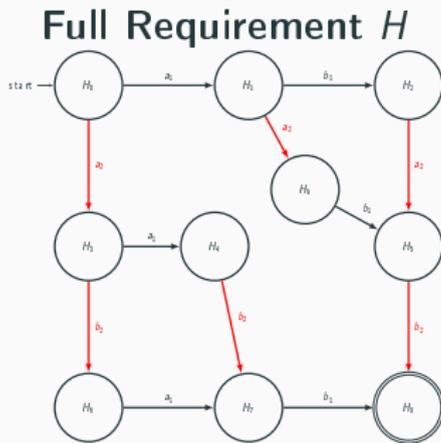
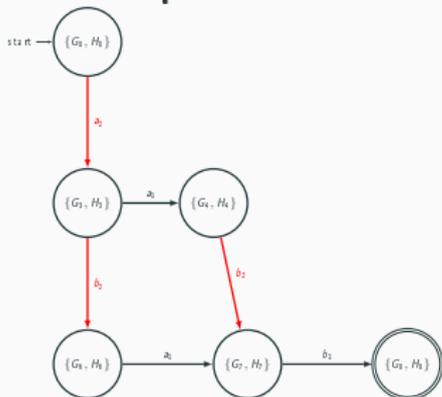


- $\{G_5, H_5\}$ is non-accessible (unreachable from the initial state $\{G_0, H_0\}$)

Example 2 - Controller Synthesis



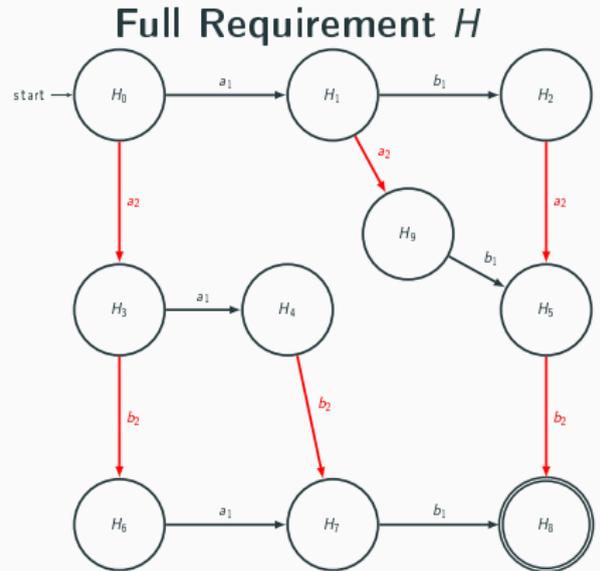
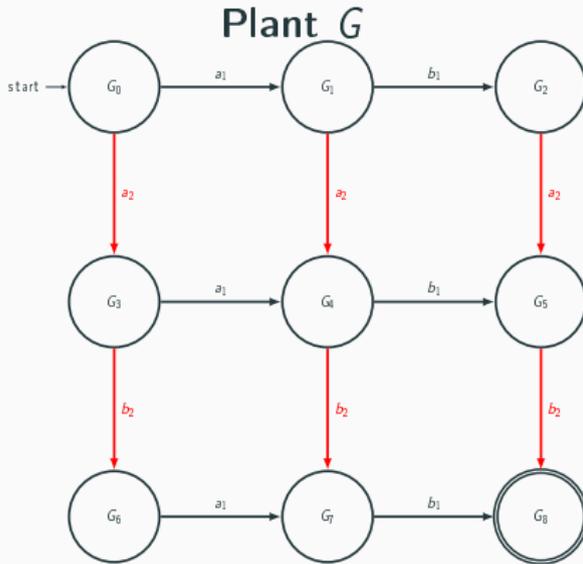
Final Supervisor S



Control Policy:

- At the beginning S disables a_1 .
- If the **plant** G is in state G_4 , S disables b_1 .

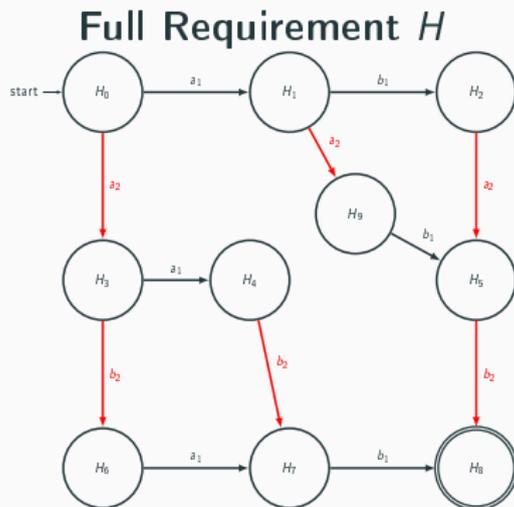
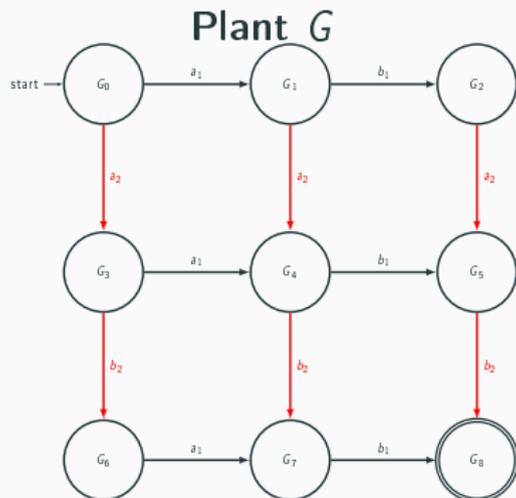
Essential Requirement



Requirement: a_1 precedes a_2 if and only if b_1 precedes b_2

Question: Can we write some other R so that $G \parallel R \equiv G \parallel H$ (i.e., such that $\mathcal{L}(G \parallel R) = \mathcal{L}(G \parallel H)$ and $\mathcal{L}_m(G \parallel R) = \mathcal{L}_m(G \parallel H)$)?

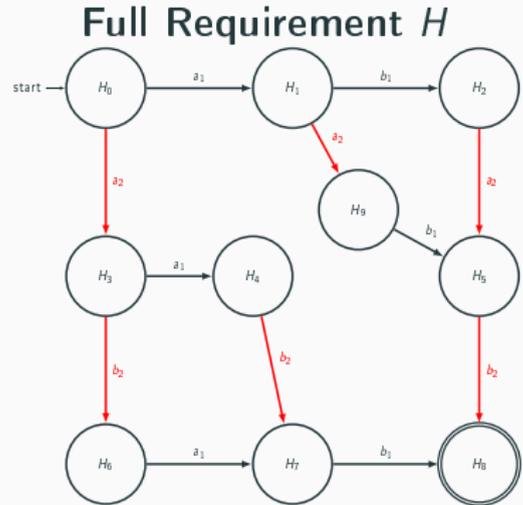
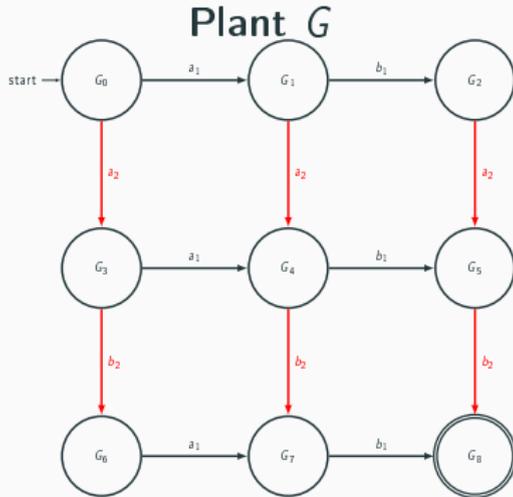
Essential Requirement - Decomposition



Requirement: $\underbrace{a_1 \text{ precedes } a_2}_A \iff \underbrace{b_1 \text{ precedes } b_2}_B$

- $A \Rightarrow B$: If a_1 precedes a_2 , then b_1 precedes b_2
- $B \Rightarrow A$: If b_1 precedes b_2 , then a_1 precedes a_2

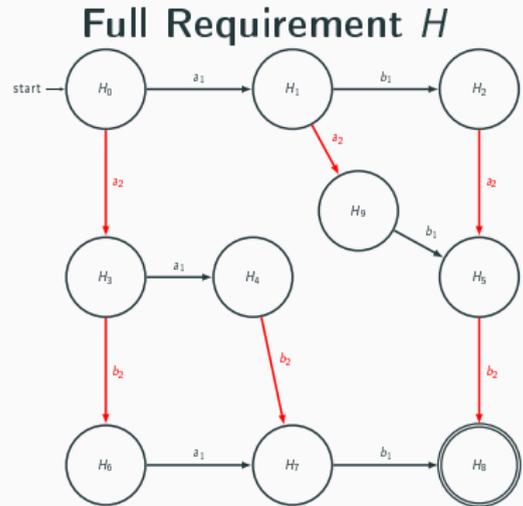
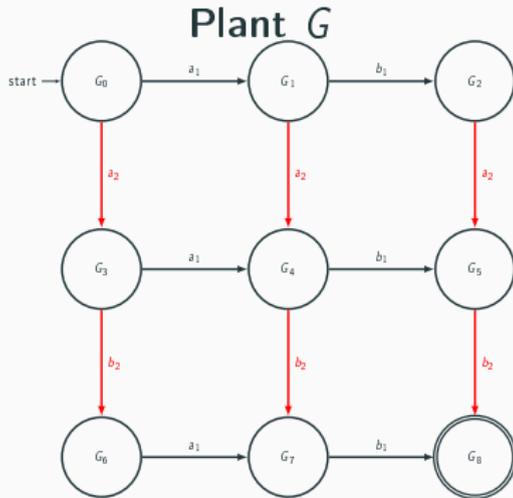
Essential Requirement - Logical Equivalence Rewriting



Requirement: $\underbrace{A}_{a_1 \text{ precedes } a_2} \Leftrightarrow \underbrace{B}_{b_1 \text{ precedes } b_2}$

- $A \Rightarrow B$: If a_1 precedes a_2 , then b_1 precedes b_2
- $B \Rightarrow A$: If ~~b_1 precedes b_2~~ , then ~~a_1 precedes a_2~~
- $\neg A \Rightarrow \neg B$: If a_1 does not precede a_2 , then b_1 does not precede b_2

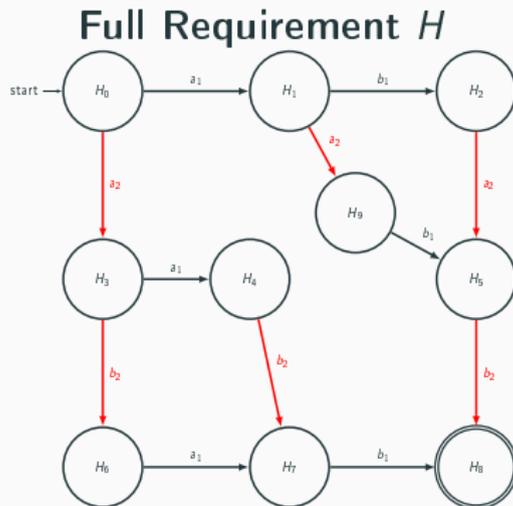
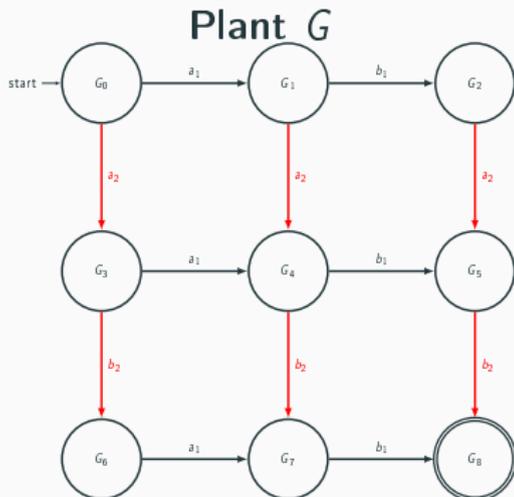
Essential Requirement - Logical Equivalence Rewriting



Requirement: $\underbrace{A}_{a_1 \text{ precedes } a_2} \iff \underbrace{B}_{b_1 \text{ precedes } b_2}$

- $A \Rightarrow B$: If a_1 precedes a_2 , then b_1 precedes b_2
- $\neg A \Rightarrow \neg B$: If ~~a_1 does not precede a_2~~ , then ~~b_1 does not precede b_2~~
- $\neg A \Rightarrow \neg B$: If a_2 precedes a_1 , then b_2 precedes b_1

Essential Requirement - Better Equivalent Decomposition

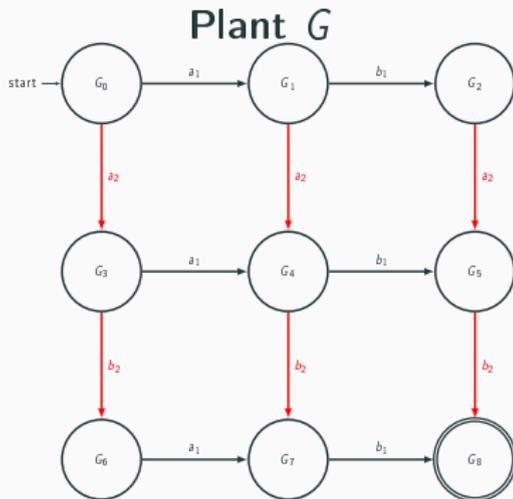


Requirement: $\underbrace{a_1 \text{ precedes } a_2}_A \iff \underbrace{b_1 \text{ precedes } b_2}_B$

• $A \Rightarrow B$: If a_1 precedes a_2 , then b_1 precedes b_2 (R₁)

• $\neg A \Rightarrow \neg B$: If a_2 precedes a_1 , then b_2 precedes b_1 (R₂)

Essential Requirement - R_1 - Attempt 1



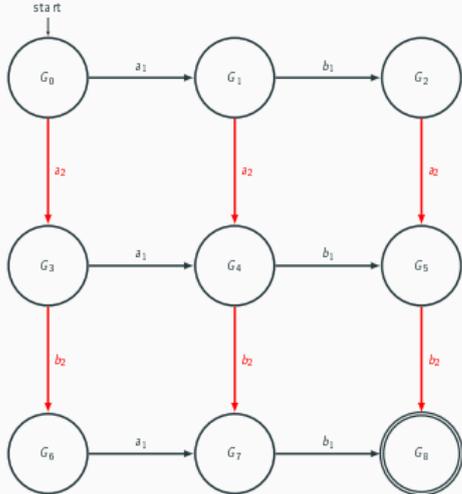
Essential Requirement R_1

- States?
- Transitions?

Requirement: $A \Rightarrow B$: If a_1 precedes a_2 , then b_1 precedes b_2

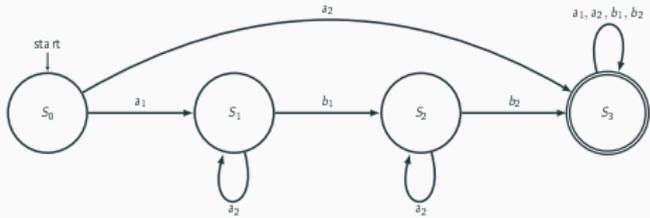
Essential Requirement - R_1 - Attempt 1

Plant G



Structure! Every path from G_0 to G_8 contains exactly 1 occurrence of each event.

Essential Requirement R_1



Rationale:

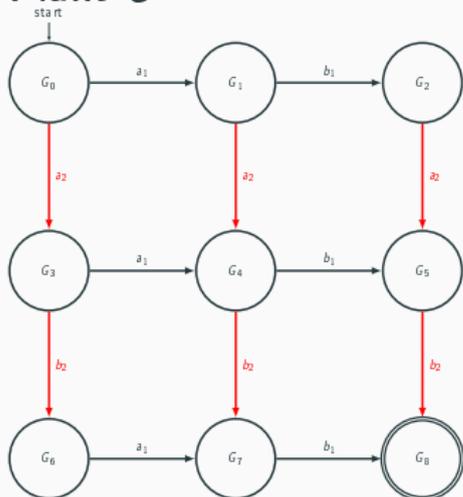
- We care about seeing either a_1 or a_2 at the beginning.
- If it's going to be a_2 , then whatever happens is ok.
- Otherwise it's going to be a_1 and the idea is that we eventually see a_2 (it's not important exactly when) and we need to see b_1 before b_2 .

Can we improve R_1 ?

Requirement R_1 : $A \Rightarrow B$: If a_1 precedes a_2 , then b_1 precedes b_2

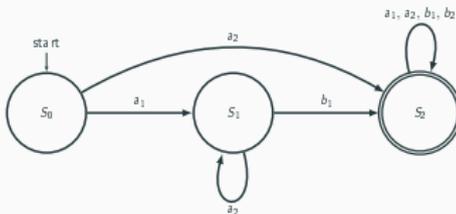
Essential Requirement - R_1 - Attempt 2

Plant G



Structure! Every path from G_0 to G_8 contains exactly 1 occurrence of each event.

Essential Requirement R_1



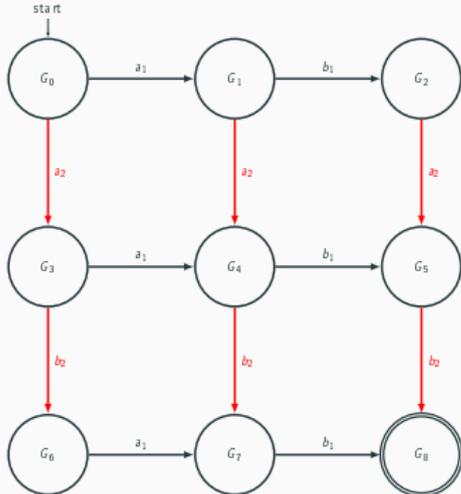
Rationale:

- We care about seeing either a_1 or a_2 at the beginning.
- If it's going to be a_2 , then whatever happens is ok.
- Otherwise it's going to be a_1 and the idea is that we eventually see a_2 (it's not important exactly when) and we need to see b_1 before b_2 .

Requirement R_1 : $A \Rightarrow B$: If a_1 precedes a_2 , then b_1 precedes b_2

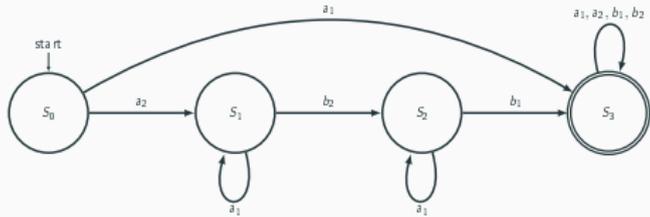
Essential Requirement - R_2 - Attempt 1

Plant G



Structure! Every path from G_0 to G_8 contains exactly 1 occurrence of each event.

Essential Requirement R_2



Rationale:

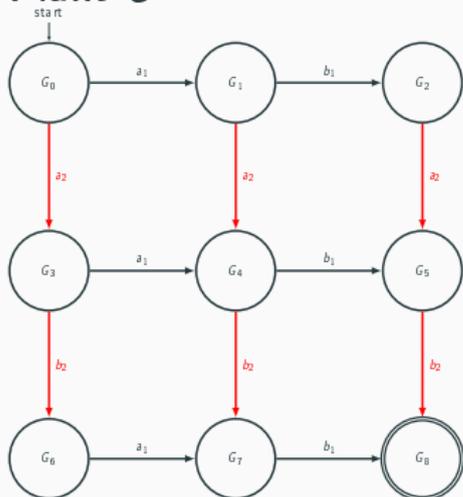
- We care about seeing either a_1 or a_2 at the beginning.
- If it's going to be a_1 , then whatever happens is ok.
- Otherwise it's going to be a_2 and the idea is that we eventually see a_1 (it's not important exactly when) and we need to see b_2 before b_1 .

Can we improve R_2 ?

Requirement R_2 : $\neg A \Rightarrow \neg B$: If a_2 precedes a_1 , then b_2 precedes b_1

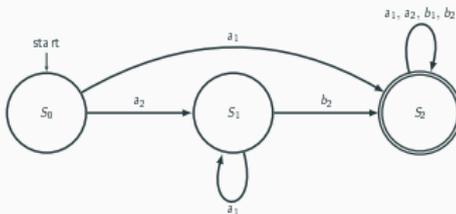
Essential Requirement - R_2 - Attempt 2

Plant G



Structure! Every path from G_0 to G_8 contains exactly 1 occurrence of each event.

Essential Requirement R_2



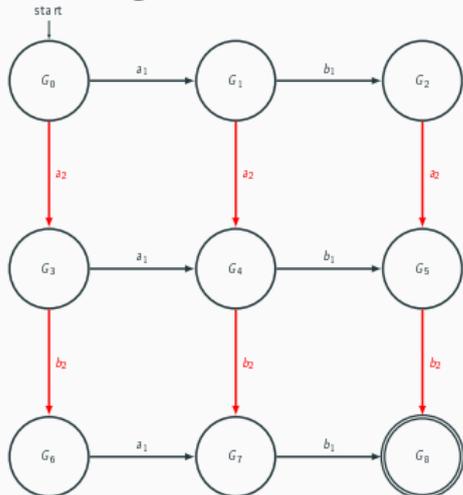
Rationale:

- We care about seeing either a_1 or a_2 at the beginning.
- If it's going to be a_1 , then whatever happens is ok.
- Otherwise it's going to be a_2 and the idea is that we eventually see a_1 (it's not important exactly when) and we need to see b_2 before b_1 .

Requirement R_2 : $\neg A \Rightarrow \neg B$: If a_2 precedes a_1 , then b_2 precedes b_1

Essential Requirement - $R := R_1 \parallel R_2$

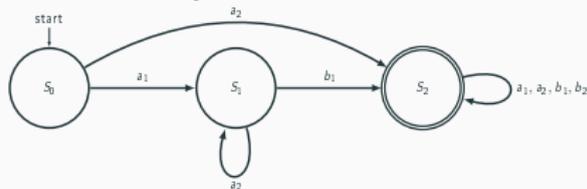
Plant G



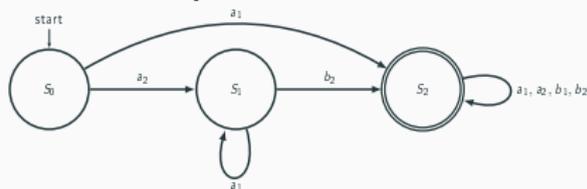
Structure! Every path from G_0 to G_8 contains exactly 1 occurrence of each event.

Requirement R : $A \Leftrightarrow B : a_2$ precedes a_1
iff b_2 precedes b_1

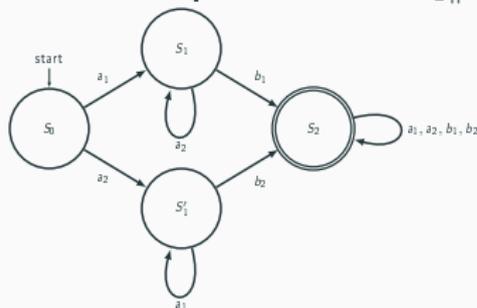
Essential Requirement R_1



Essential Requirement R_2

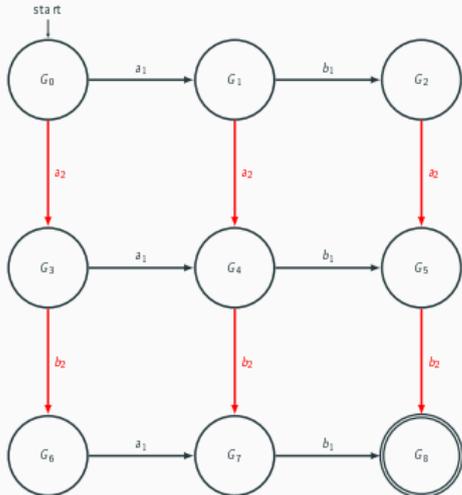


Essential Requirement $R := R_1 \parallel R_2$



Essential Requirement - R

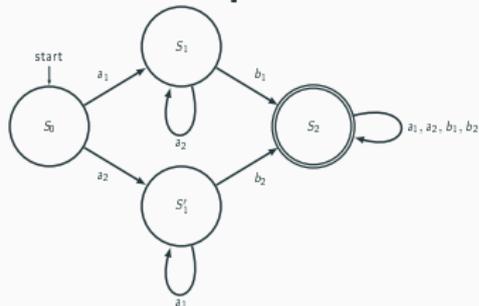
Plant G



Structure! Every path from G_0 to G_8 contains exactly 1 occurrence of each event.

Requirement R : $A \Leftrightarrow B : a_2$
precedes a_1 iff b_2 precedes b_1

Essential Requirement R



Full Requirement $H := G \parallel R$

