Systems Design Laboratory

Eclipse Supervisory Control Engineering Toolkit (ESCET)

Matteo Zavatteri

[1]Department of Mathematics, University of Padova, ITALY

[2]Department of Computer Science, University of Verona, ITALY

Motivation for Model-Based Systems Engineering

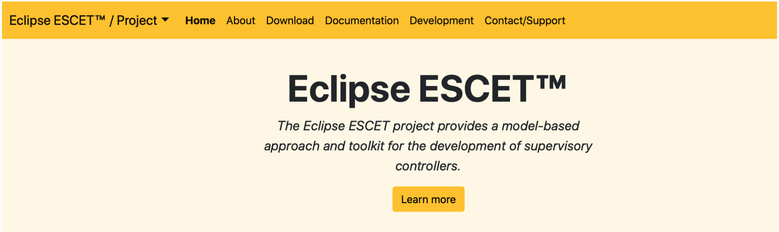
In the development of systems and supervisory controllers:

- 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 Supervisory Control Engineering Toolkit

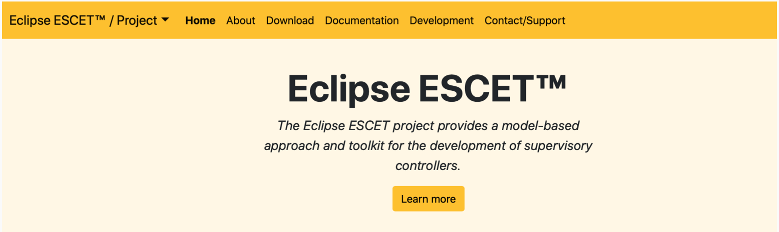


The Eclipse Supervisory Control Engineering Toolkit (Eclipse ESCET™) project is an Eclipse Foundation open-source project that provides a toolkit for the development of supervisory controllers in the Model-Based Systems Engineering (MBSE) paradigm.

- **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.**

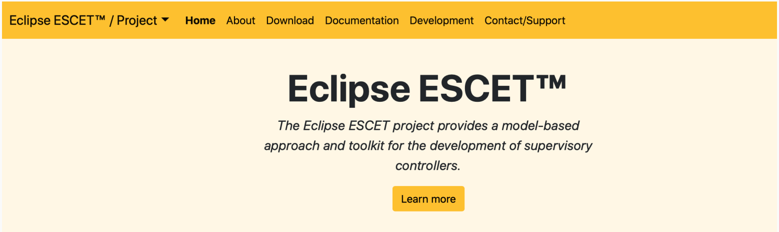
https://www.eclipse.org/escet/

Model-Based Systems Engineering



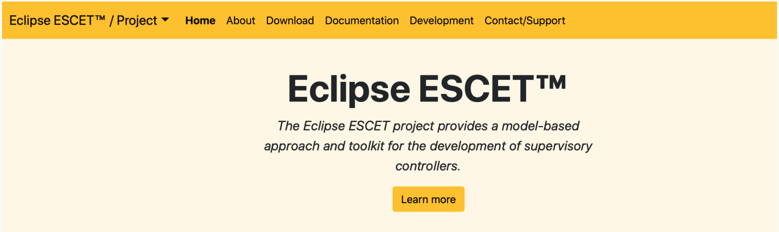
- The toolkit has a strong focus on model-based design, supervisory controller synthesis, and industrial applicability, for example to cyber-physical systems.
- The toolkit supports the entire development process of (supervisory) controllers, from modeling, supervisory controller synthesis, simulation-based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

Eclipse Supervisory Control Engineering Toolkit (ESCET)



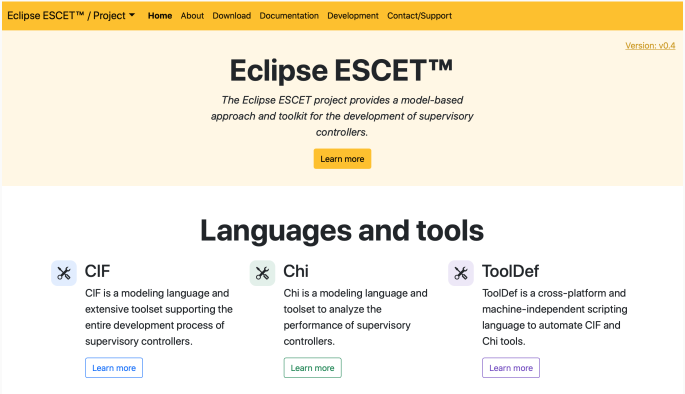
The Eclipse Supervisory Control Engineering Toolkit (ESCET) was developed approximately over a period of approximately two decades (starting from the early 2000s) at the Eindhoven University of Technology (TU/e) in cooperation with many European and national projects.

Eclipse Supervisory Control Engineering Toolkit (ESCET)



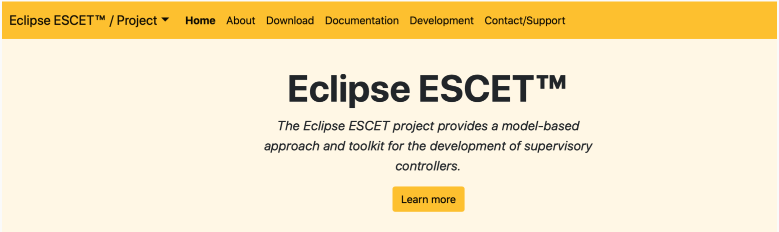
In 2021, Eclipse ESCET became an independent Eclipse Foundation open source project, and is no longer formally associated with the TU/e.

Eclipse Supervisory Control Engineering Toolkit (ESCET)



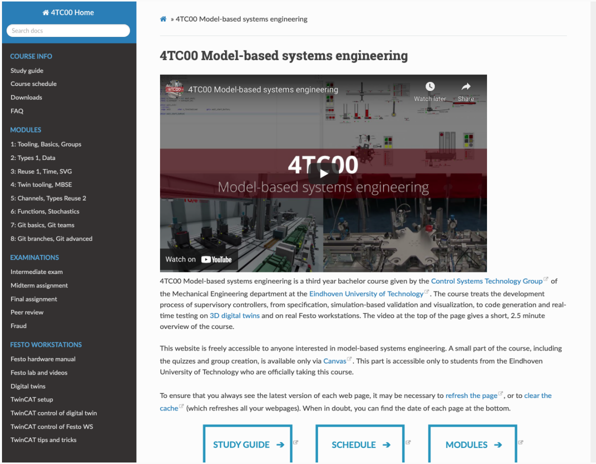
Eclipse ESCET is based on CIF: the *Compositional Interchange Format* for hybrid systems. CIF is an automata-based modeling language for the specification of discrete event, timed, and hybrid systems.

Eclipse Supervisory Control Engineering Toolkit (ESCET)



- 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

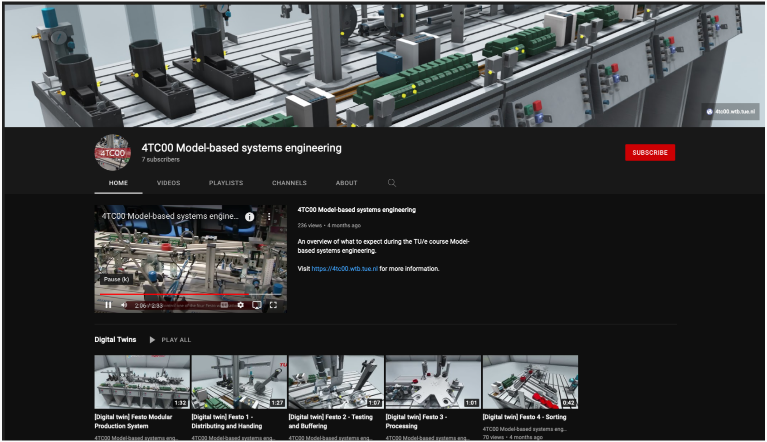
Eindhoven University of Technology (TU/e) - 4TC00 course



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

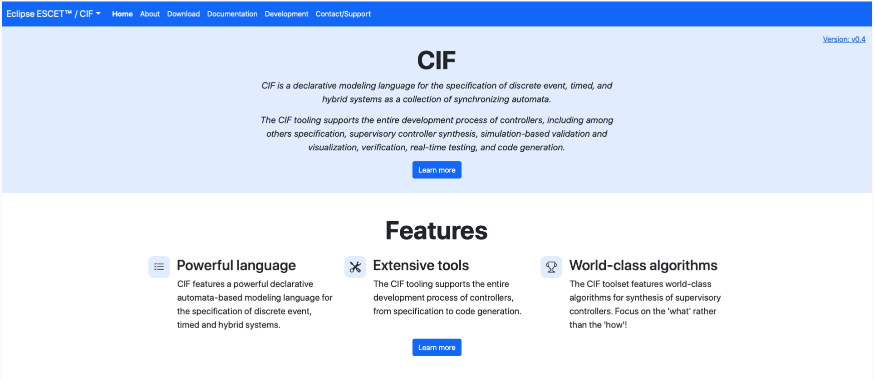
Eindhoven University of Technology (TU/e) - 4TC00 course



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

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

Compositional Interchange Format (CIF)



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

CIF - Language

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

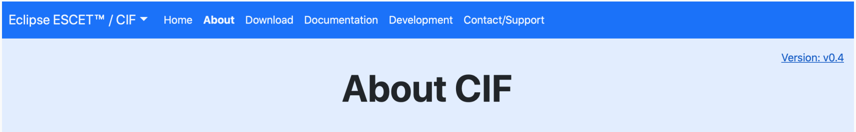
Version: v0.4

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.

CIF - Tools

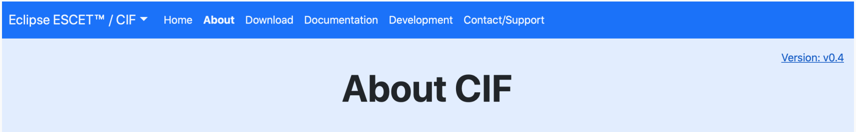


The CIF tooling supports the entire development process of controllers, including among others specification, supervisory controller synthesis, simulation-based validation and visualization, verification, real-time testing, and code generation.

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.

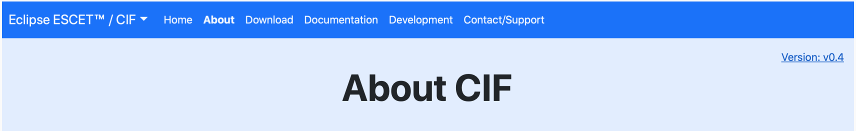
CIF - Supervisory controller synthesis



Supervisory controller synthesis is a key feature of CIF.

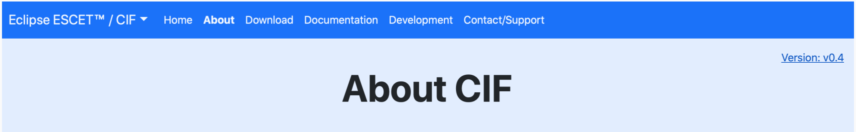
- It involves the automatic generation of supervisory controllers from a specification of the uncontrolled system and the (safety) requirements that the controller needs to enforce.
- This moves controller design from "how should the implementation work" to "what should the controller do".
- Implementation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

CIF - Application



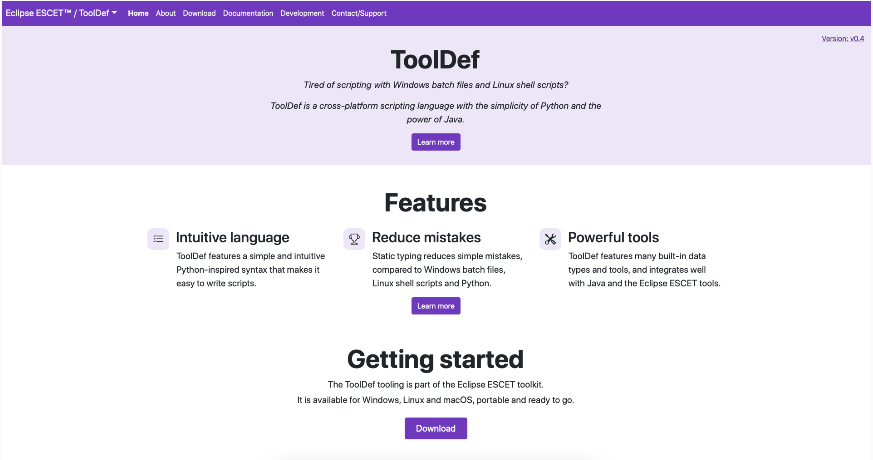
- CIF has been applied in industry, for various domains, including the medical, semiconductor and public works (infrastructure) domains.
- The main application area of CIF is the development of supervisory controllers.
- The language and tools are generic, and can be used to work with state machines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

CIF - Application



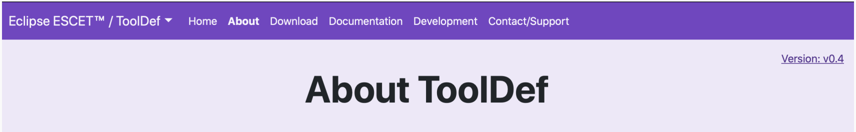
- The CIF language and tools are being developed as part of the Eclipse ESCET open-source project.
- The CIF tools are part of the Eclipse ESCET toolkit.

ToolDef: An Integrated Scripting Language



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

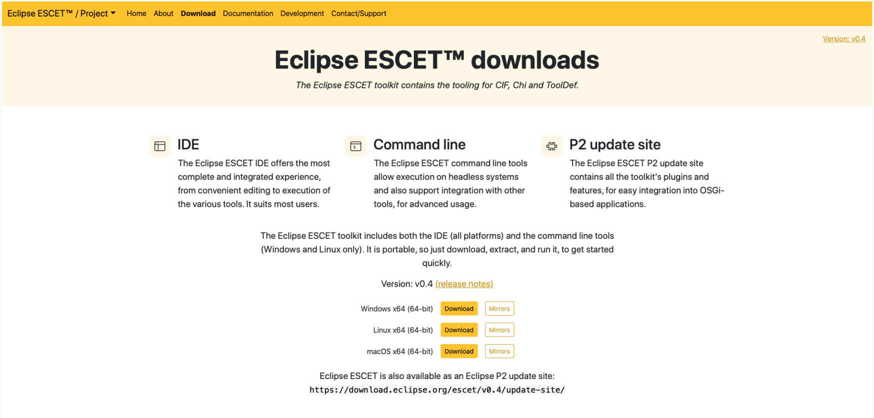
ToolDef - Language



ToolDef allows us to:

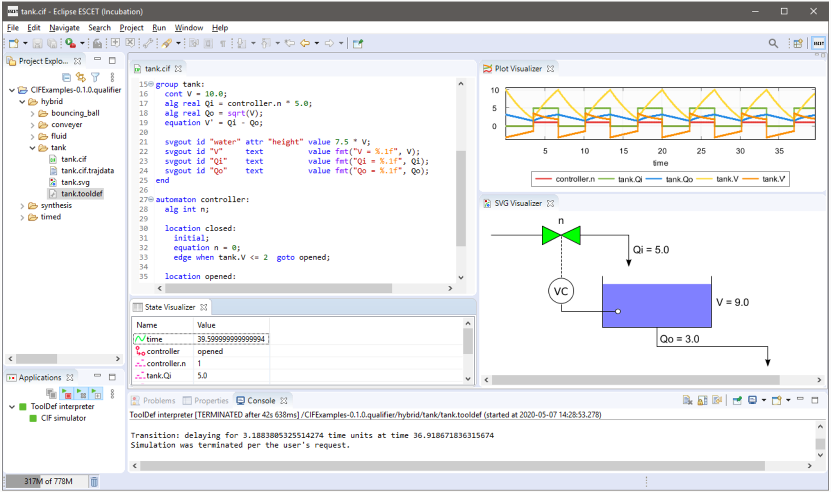
- write scripts using a simple and intuitive syntax, loosely based on the better aspects of Python.
- catch simple mistakes early on due to static typing.
- work with data of all kinds, using a large number of built-in data types.
- manipulate data and paths, work with files and directories, and much 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 Supervisory Control Engineering Toolkit (ESCET)



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

ESCET - Integrated Development Environment



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

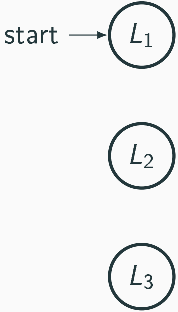
Locations can be:

- **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

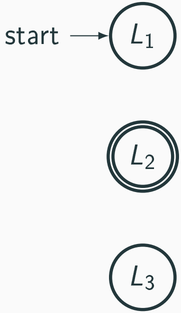
Locations can be:

- **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

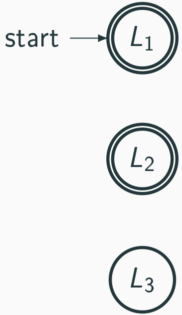
Locations can be:

- **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.

CIF - Events

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.

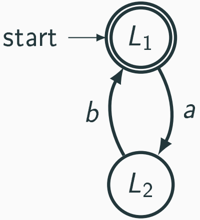
CIF - Edges (i.e., Transitions)

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,...`

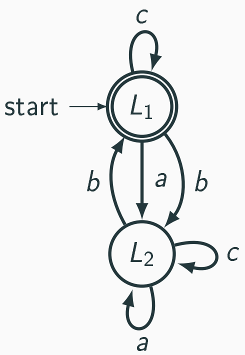
CIF - Edges (cont.)

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

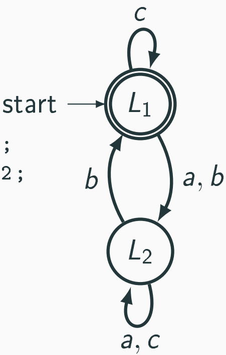
```
event a;
automaton Component:
    event b;
    event c;
    location L1:
      initial; marked;
      edge a goto L2;
      edge b goto L2;
      edge c goto L1;

    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:

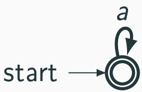
```
edge a[,b,...] [goto Lj];
```

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

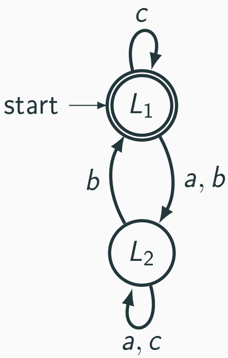
Automaton Alphabet:

- **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

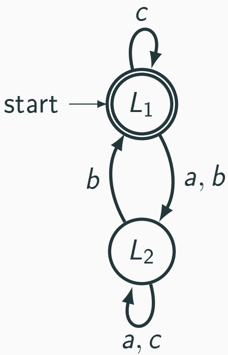
Automaton Alphabet:

- 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`.

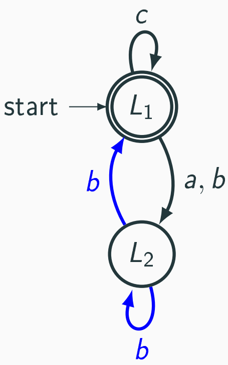
CIF - Non Determinism

- 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

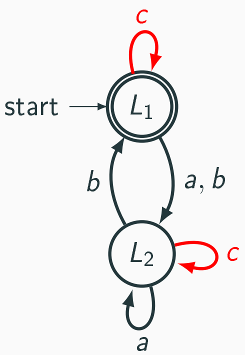
```
uncontrollable event a[,b,...];
uncontrollable a[,b,...];
```

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];
```

Supervisory Control - Automata Types

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
```

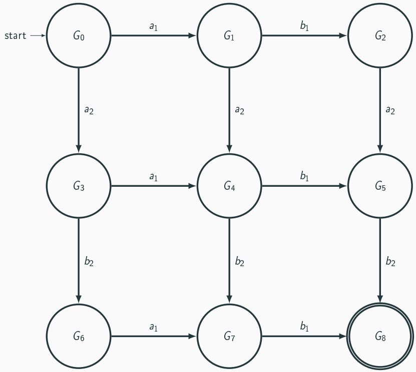
$\Rightarrow$

```
plant C:
  ...
end

requirement R:
  ...
end

supervisor S:
  ...
end
```

The Database Concurrency Example



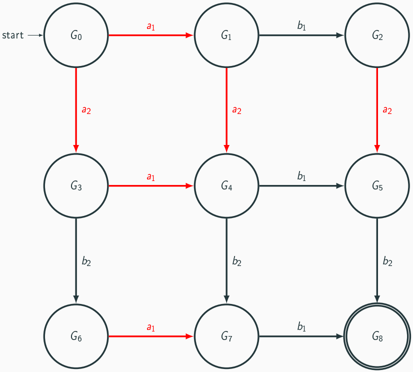
- Two transactions
  - $T_1 := a_1 b_1$
  - $T_2 := a_2 b_2$

  (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

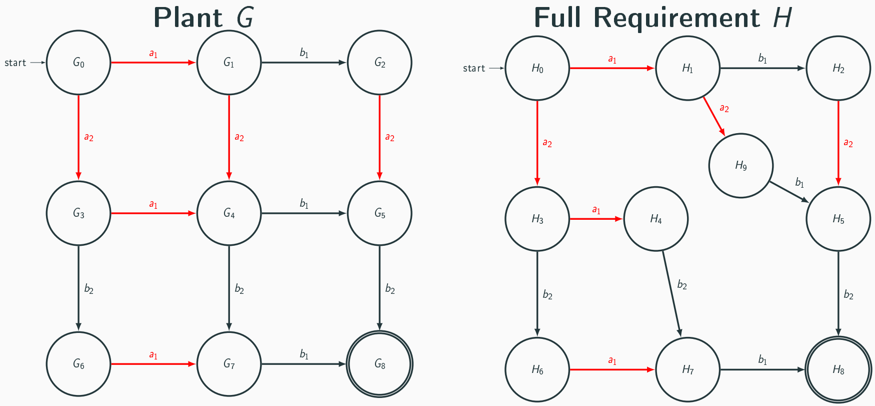
Example 1



- 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

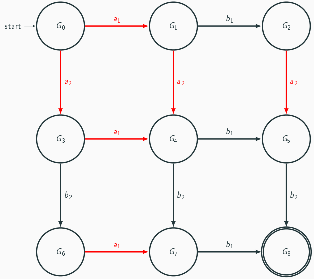
Example 1 - Plant and Requirement



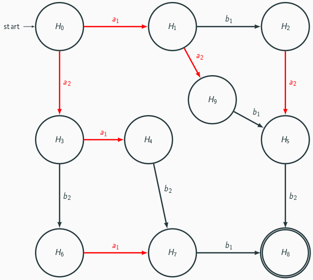
Requirement: a_1 **precedes** a_2 **if and only if** b_1 **precedes** b_2

Note: Full Requirement := Plant || Essential Requirement

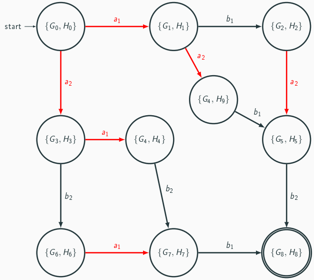
Plant



Requirement



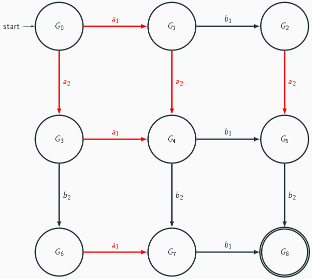
Tentative Supervisor $G \| H$



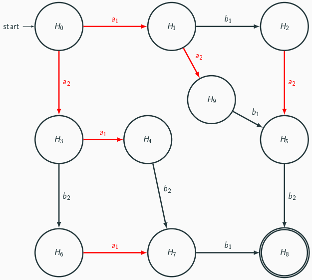
Requirement: a_1 precedes a_2 if and only if b_1 precedes b_2

Example 1 - Controller Synthesis

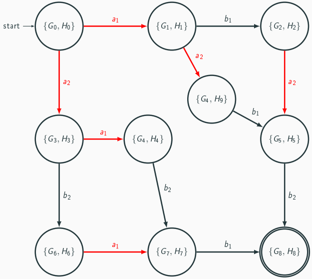
Plant G



Full Requirement H



Supervisor S



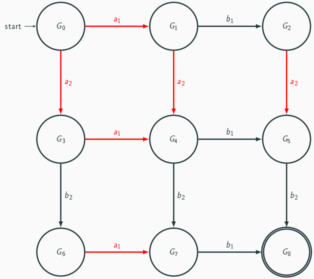
Requirement: a_1 precedes a_2 if and only if b_1 precedes b_2

- No states to remove

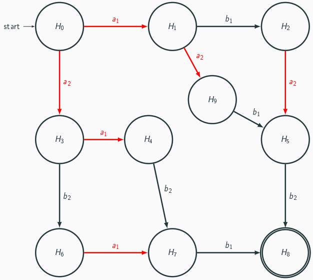
$$\Downarrow$$

- Final supervisor

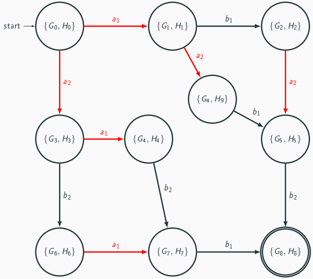
Plant G



Full Requirement H



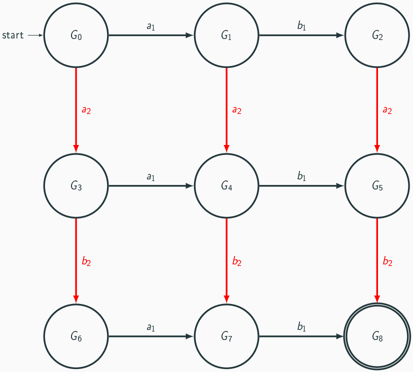
Supervisor S



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

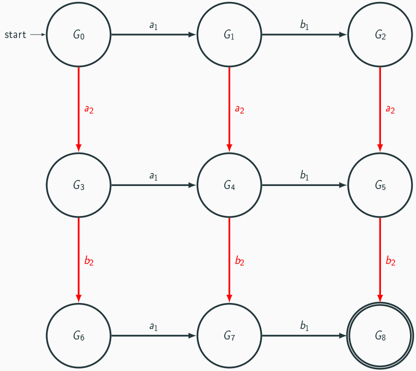


- Events a_1, b_1 are controllable
- Events a_2, b_2 are uncontrollable
- G_0 is the initial state
- G_8 is the marked state

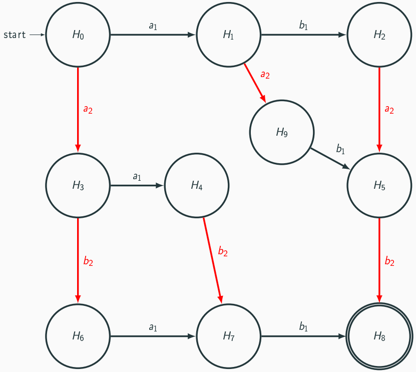
Same Requirement: a_1 **precedes** a_2 **if and only if** b_1 **precedes** b_2

Example 2 - Plant and Requirement

Plant G

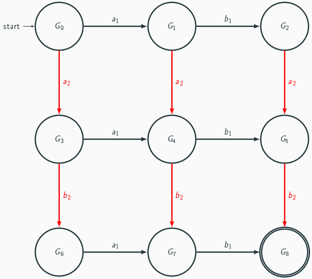


Full Requirement H

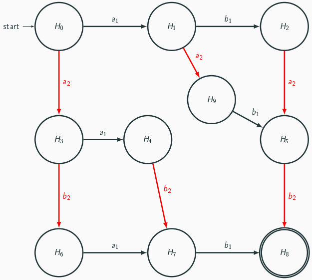


Example 2 - Controller Synthesis

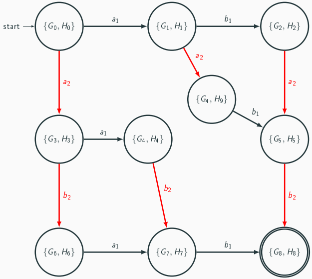
Plant G



Full Requirement H



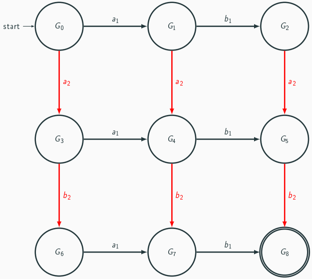
Tentative Supervisor $G \| H$



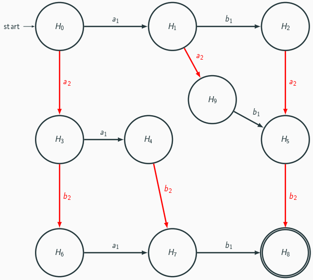
- Any problems?

Example 2 - Controller Synthesis

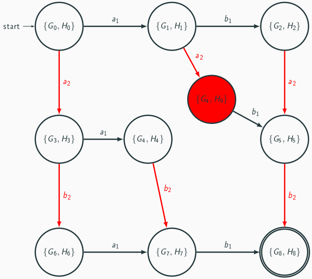
Plant G



Full Requirement H



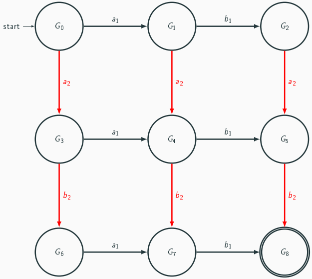
Tentative Supervisor $G \| H$

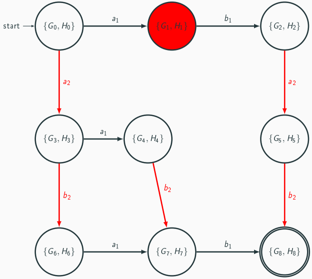


- $\{G_4, H_9\}$ is uncontrollable

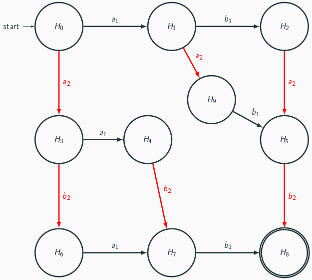
Example 2 - Controller Synthesis

Plant G



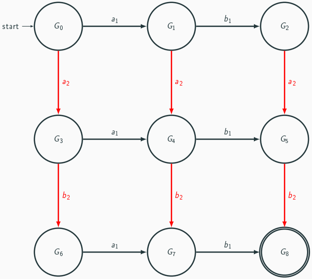


Full Requirement H

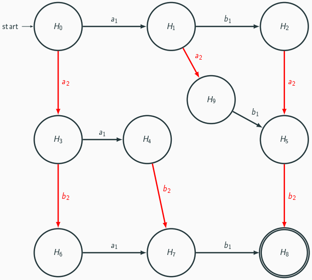


- $\{G_1, H_1\}$ is uncontrollable

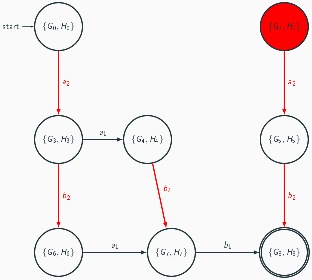
Plant G



Full Requirement H



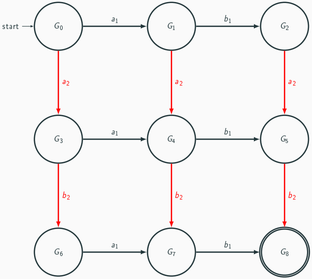
Tentative Supervisor $G \| H$



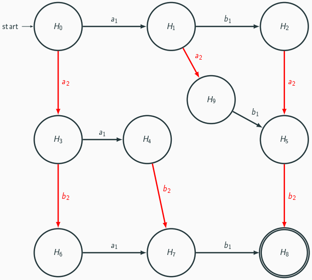
- $\{G_2, H_2\}$ is non-accessible (unreachable from the initial state $\{G_0, H_0\}$)

Example 2 - Controller Synthesis

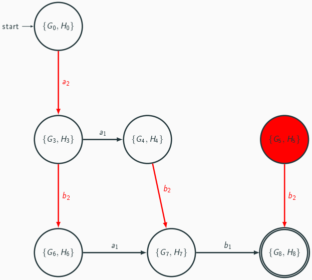
Plant G



Full Requirement H



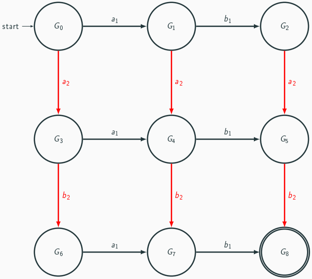
Tentative Supervisor $G \| H$



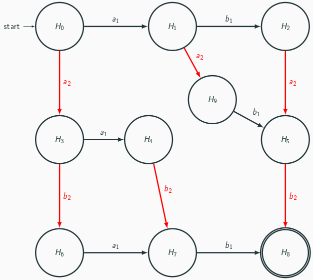
- $\{G_5, H_5\}$ is non-accessible (unreachable from the initial state $\{G_0, H_0\}$)

Example 2 - Controller Synthesis

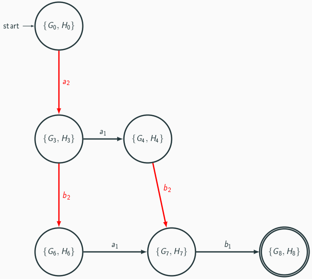
Plant G



Full Requirement H



Final Supervisor S

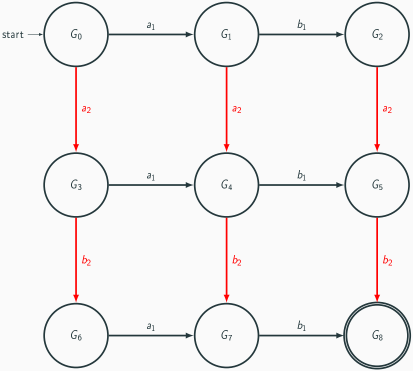


Control Policy:

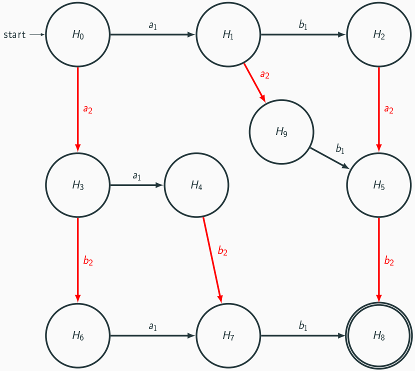
- At the beginning S disables a_1.
- If the **plant** G is in state G_4, S disables b_1.

Essential Requirement

Plant G



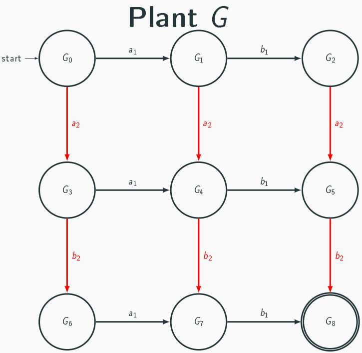
Full Requirement H

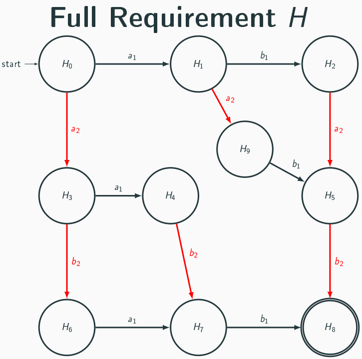


Requirement: a_1 **precedes** a_2 **if and only if** b_1 **precedes** b_2

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

Essential Requirement - Decomposition



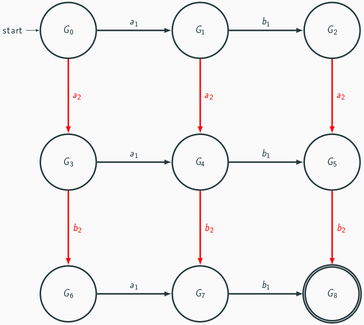


Requirement: $\overbrace{a_1 \textbf{ precedes } a_2}^{A}\ \overbrace{\textbf{if and only if}}^{\Leftrightarrow}\ \overbrace{b_1 \textbf{ precedes } b_2}^{B}$

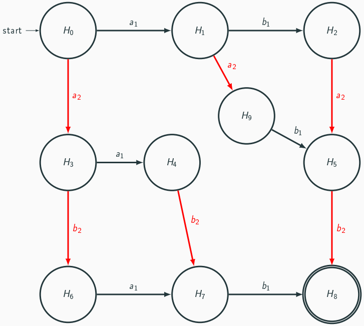
- $A \Rightarrow B$: **If** a_1 **precedes** a_2**, then** b_1 **precedes** b_2
- $B \Rightarrow A$: **If** b_1 **precedes** b_2**, then** a_1 **precedes** a_2

Essential Requirement - Logical Equivalence Rewriting

Plant G



Full Requirement H

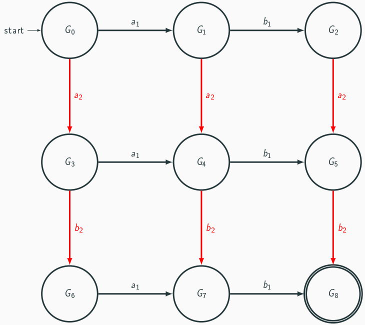


Requirement: $\overbrace{a_1 \text{ precedes } a_2}^{A}\ \overbrace{\text{if and only if}}^{\Leftrightarrow}\ \overbrace{b_1 \text{ precedes } b_2}^{B}$

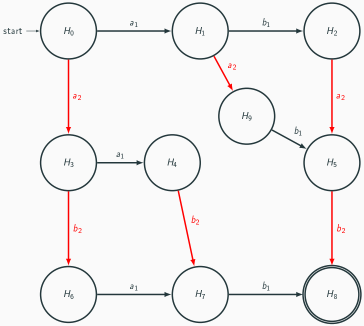
- $A \Rightarrow B$: **If** a_1 **precedes** a_2**, then** b_1 **precedes** b_2
- ~~$B \Rightarrow A$: **If** b_1 **precedes** b_2**, then** a_1 **precedes** a_2~~
- $\neg A \Rightarrow \neg B$: **If** a_1 **does not precede** a_2**, then** b_1 **does not precede** b_2

Essential Requirement - Logical Equivalence Rewriting

Plant G



Full Requirement H

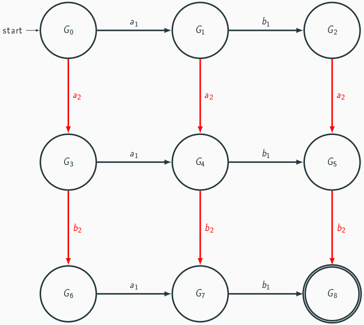


Requirement: $\overbrace{a_1 \text{ precedes } a_2}^{A}$ $\overbrace{\text{if and only if}}^{\Leftrightarrow}$ $\overbrace{b_1 \text{ precedes } b_2}^{B}$

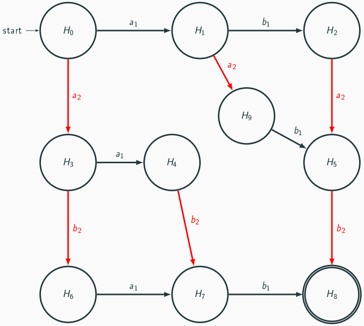
- $A \Rightarrow B$: **If a_1 precedes a_2, then b_1 precedes b_2**
- ~~$\neg A \Rightarrow \neg B$: **If a_1 does not precede a_2, then b_1 does not precede b_2**~~
- $\neg A \Rightarrow \neg B$: **If a_2 precedes a_1, then b_2 precedes b_1**

Essential Requirement - Better Equivalent Decomposition

Plant G



Full Requirement H

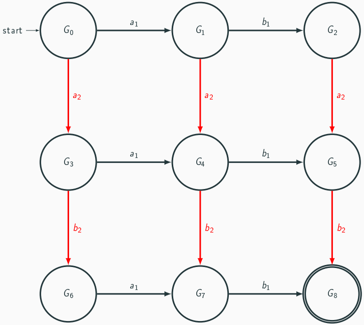


Requirement: $\overbrace{a_1 \text{ precedes } a_2}^{A}\ \overbrace{\text{if and only if}}^{\Leftrightarrow}\ \overbrace{b_1 \text{ precedes } b_2}^{B}$

- $A \Rightarrow B$: **If a_1 precedes a_2, then b_1 precedes b_2** (R_1)
- $\neg A \Rightarrow \neg B$: **If a_2 precedes a_1, then b_2 precedes b_1** (R_2)

Essential Requirement - R_1 - Attempt 1

Plant G



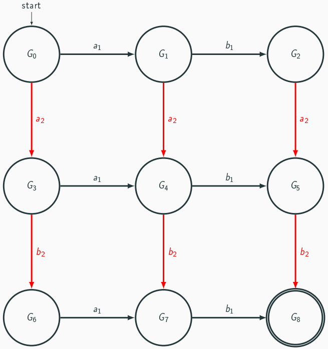
Essential Requirement R_1

- States?
- Transitions?

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

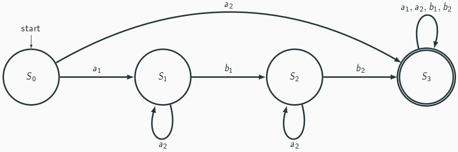
Essential Requirement - R_1 - Attempt 1

Plant G



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

Essential Requirement R_1



Rationale:

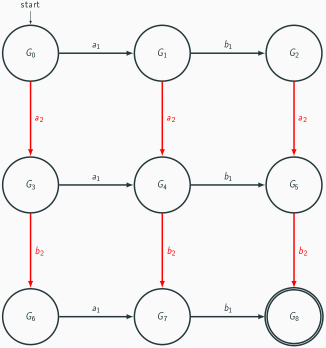
- **We care about seeing either a_1 or a_2 at the beginning.**
- **If it's going to be a_2, then whatever happens is ok.**
- **Otherwise it's going to be a_1 and the idea is that we eventually see a_2 (it's not important exactly when) and we need to see b_1 before b_2.**

Can we improve R_1?

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

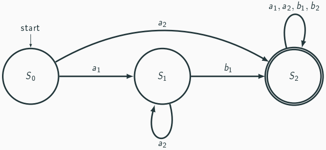
Essential Requirement - R_1 - Attempt 2

Plant G



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

Essential Requirement R_1



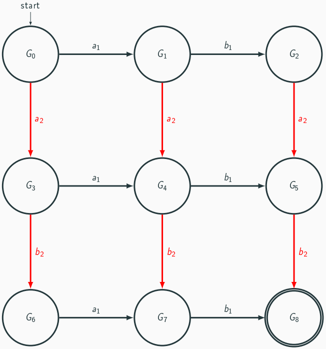
Rationale:

- **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

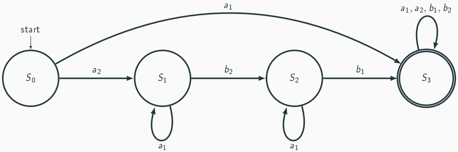
Essential Requirement - R_2 - Attempt 1

Plant G



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

Essential Requirement R_2



Rationale:

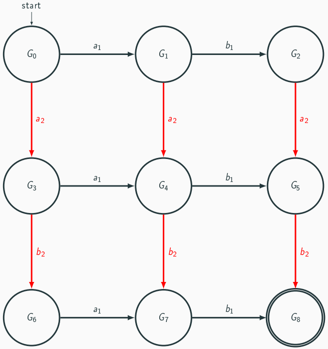
- We care about seeing either a_1 or a_2 at the beginning.
- If it's going to be a_1, then whatever happens is ok.
- Otherwise it's going to be a_2 and the idea is that we eventually see a_1 (it's not important exactly when) and we need to see b_2 before b_1.

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

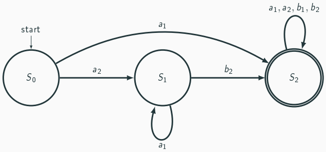
Essential Requirement - R_2 - Attempt 2

Plant G



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

Essential Requirement R_2



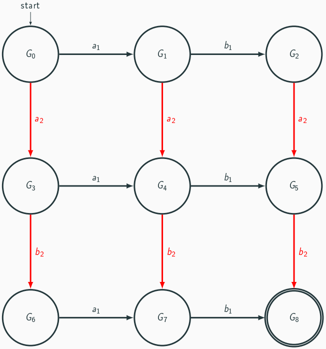
Rationale:

- **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 - $R := R_1 \| R_2$

Plant G

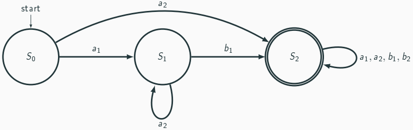


Structure! Every path from G_0 to G_8 contains exactly 1 occurrence of each event.

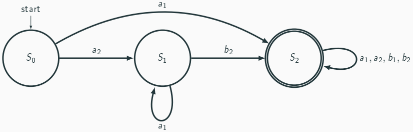
Requirement R: $A \Leftrightarrow B$: a_2 precedes a_1

iff b_2 precedes b_1

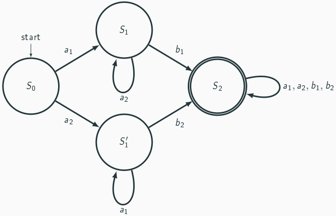
Essential Requirement R_1



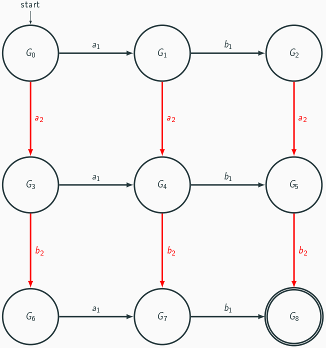
Essential Requirement R_2



Essential Requirement $R := R_1 \| R_2$



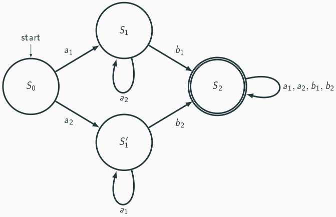
Plant G



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



Full Requirement $H := G \| R$