Bootcamp for Supervisory Control of Discrete-Event Systems

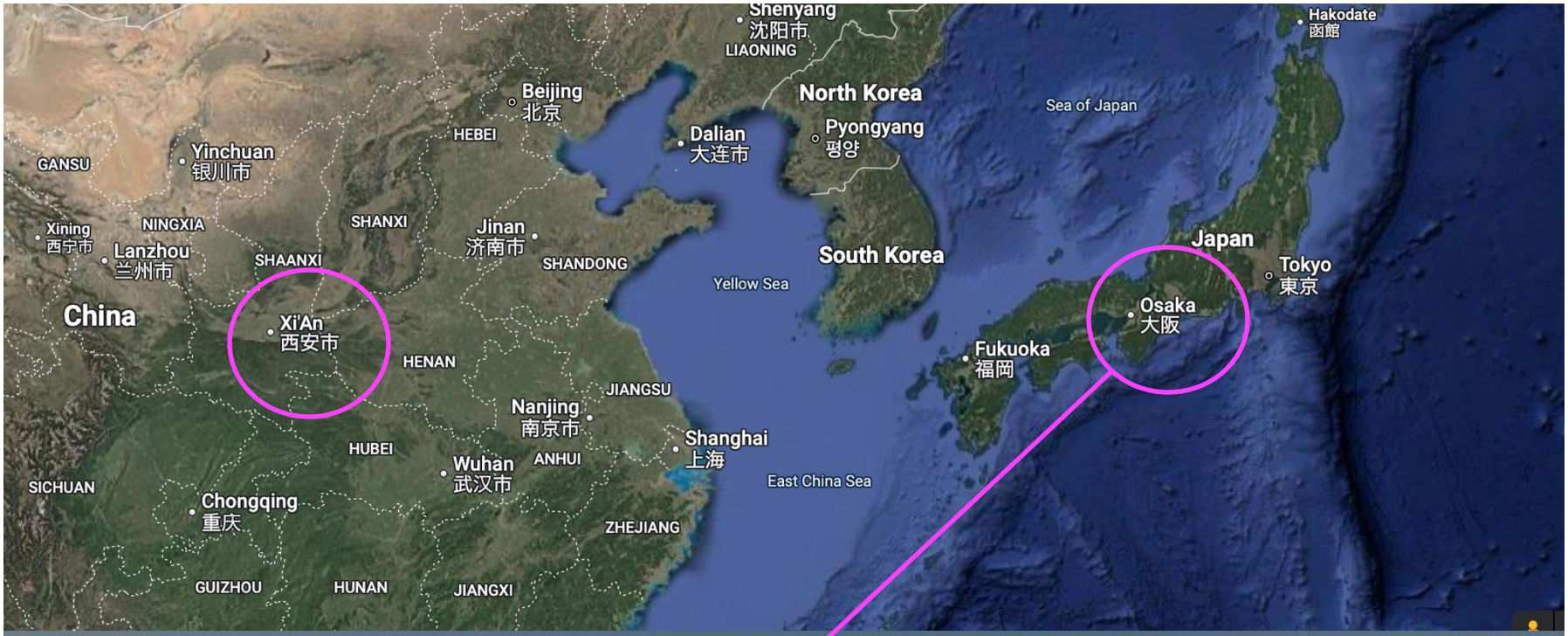
with Hands-On Python Software Tool

Kai Cai

caikai.org

Osaka Metropolitan University

Xi'an, 2024.11.07



OMU in Numbers

16,000
Full-time Students

1,400
Full-time Faculty Members

12
Undergraduate Schools

15
Graduate Schools

| 現代システム科学研究科 | 文学研究科 | 法学研究科 | 法学研究科 法曹養成専攻（法科大学院） | 経済学研究科 |

| 経営学研究科 | 都市経営研究科（社会人大学院） | 情報学研究科 | 理学研究科 | 工学研究科 | 農学研究科 |

| 獣医学研究科 | 医学研究科 | リハビリテーション学研究科 | 看護学研究科 | 生活科学研究科 |

| 創薬科学研究科（仮称・設置構想中） |

PhD program

- All in English
- Scholarship opportunities (100,000~20,000JPY/month)
- Two calls per year

Application Deadline	Onine Interview	Program Start
1. early December	mid February	April 1
2. early June	late August	October 1

Questions

- Supervisory control theory?

  → this talk

- Control theory?

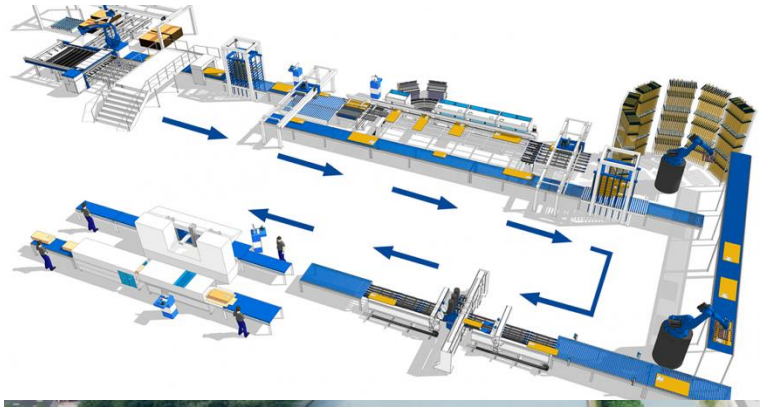
  → similarities & differences

- Murray Wonham?

  → creator

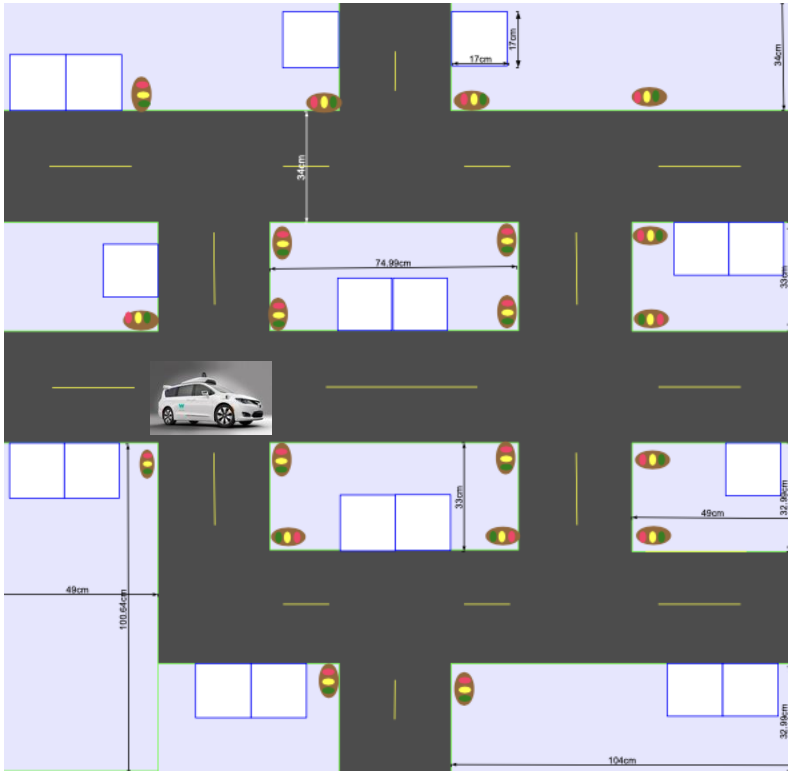
What is discrete-event systems (DES)?

- Dynamic system with
  - Discrete state set
  - Discrete and event-driven state transitions
- Examples:



Why Supervisory Control of DES?

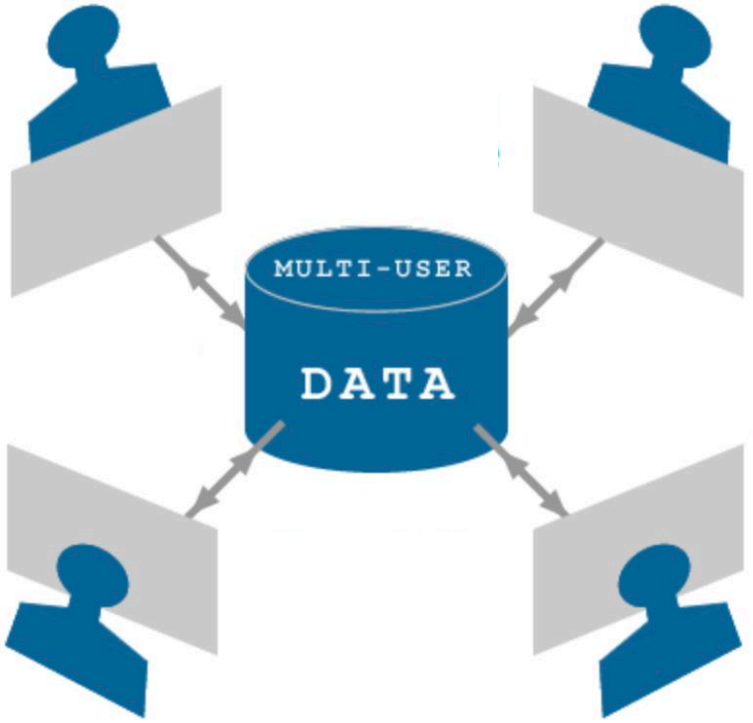
- Hybrid control
- Computers/networks
- Multi-agent path planning



$$\dot{x} = v\sin(\theta)$$
$$\dot{y} = v\cos(\theta)$$
$$\dot{\theta} = \omega$$

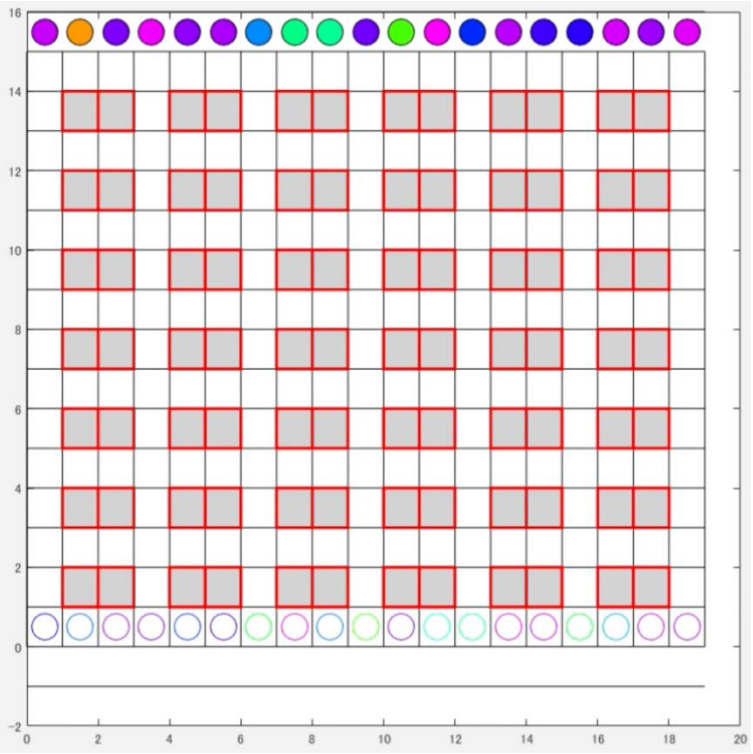
Why Supervisory Control of DES?

- Hybrid control
- Computer/networks
- Multi-agent path planning



Why Supervisory Control of DES?

- Hybrid control
- Computer/networks
- Multi-agent path planning



Content

1. Create an automaton
2. Properties of automaton
3. Synchronous product of automata
4. Feedback control loop
5. Controllability
6. Optimal supervisory control design

DES model

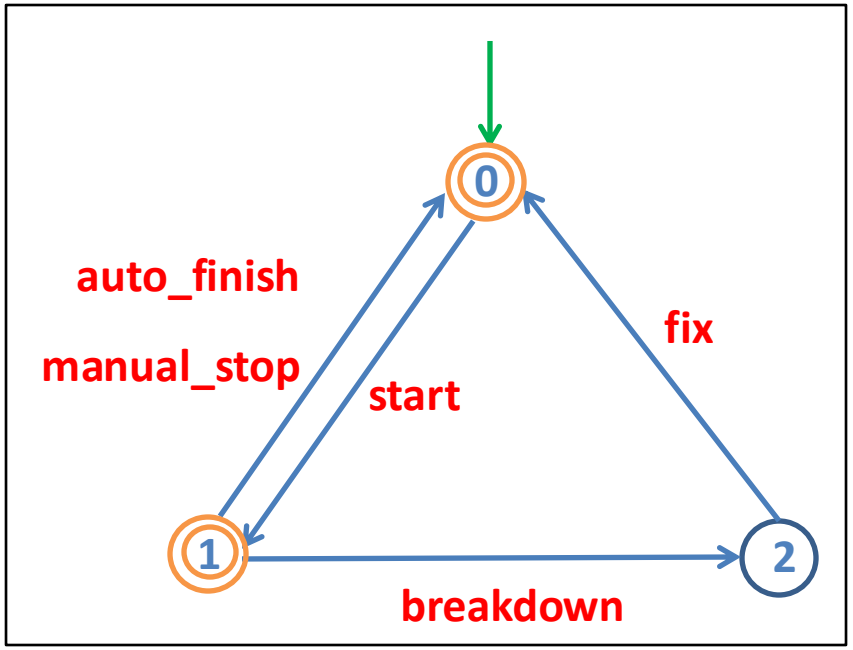
DES control

Create an automaton

A printer

States:

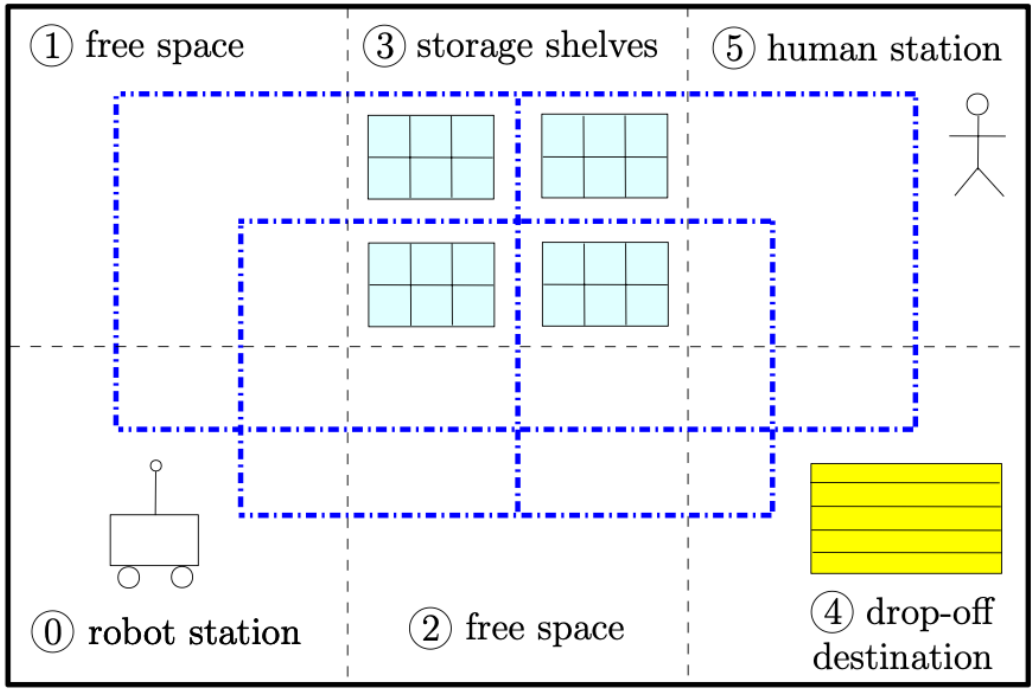
- **IDLE: 0** (initial state) (marker state)
- **WORKING: 1** (marker state)
- **BROKEN: 2**

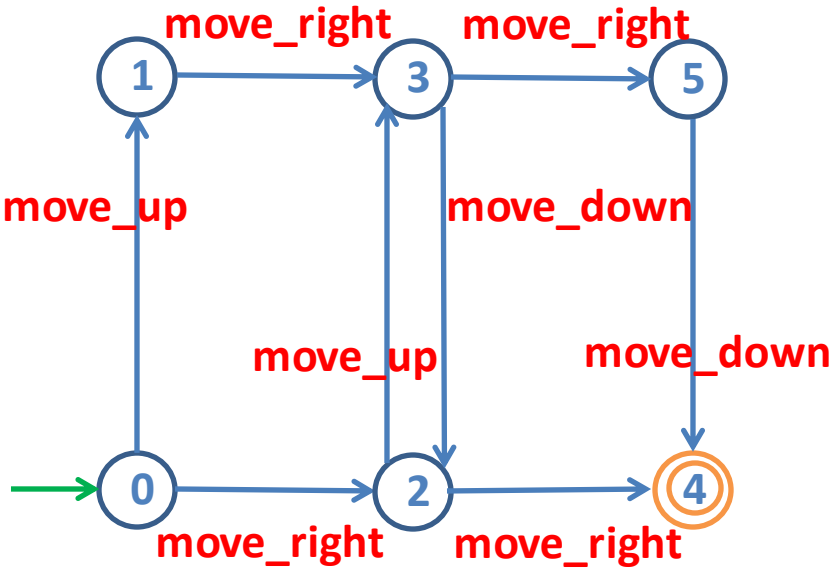


Events:

- **start (send printing job)**
- **auto_finish (printing job finished successfully)**
- **manual_stop (stop printing due to unintended problem)**
- **breakdown (paper jam)**
- **fix (jammed paper removed)**

A warehouse robot





DES model: automaton

A finite-state automaton $\mathbf{G}$ is a five tuple
$\mathbf{G} = (Q, \Sigma, \delta, q_0, Q_m)$, where

Q : finite state set (states in Q: locations or activities having duration in time)

Σ : finite event set (events in Σ: ocurring instantaneously)

δ : finite transition set

$$\delta \subseteq \{(q, \sigma, q') \mid q, q' \in Q, \sigma \in \Sigma\}$$

$q_0 \in Q$: initial state

$Q_m \subseteq Q$: marker state set

A printer

$$\mathbf{PRINTER} = (Q, \Sigma, \delta, q_0, Q_m)$$

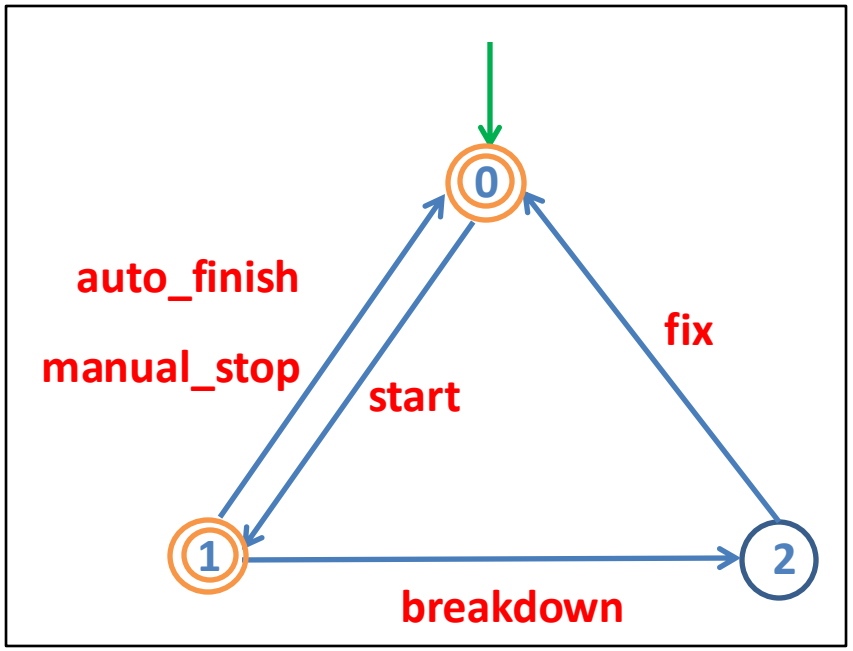
Q

q_0

Q_m

Σ

δ



A warehouse robot

$$\mathbf{ROBOT} = (Q, \Sigma, \delta, q_0, Q_m)$$

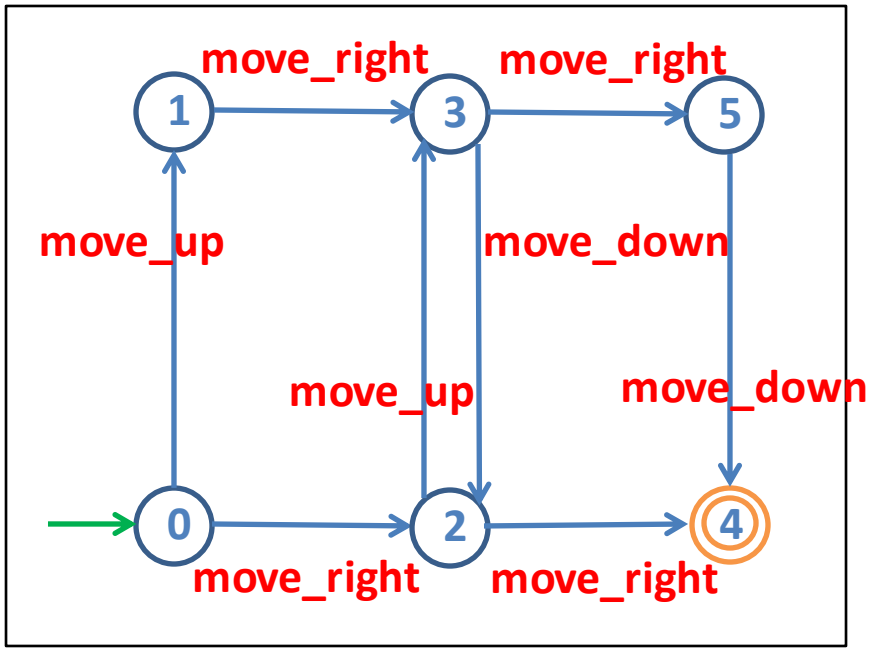
Q

q_0

Q_m

Σ

δ



PyTCT

PyTCT

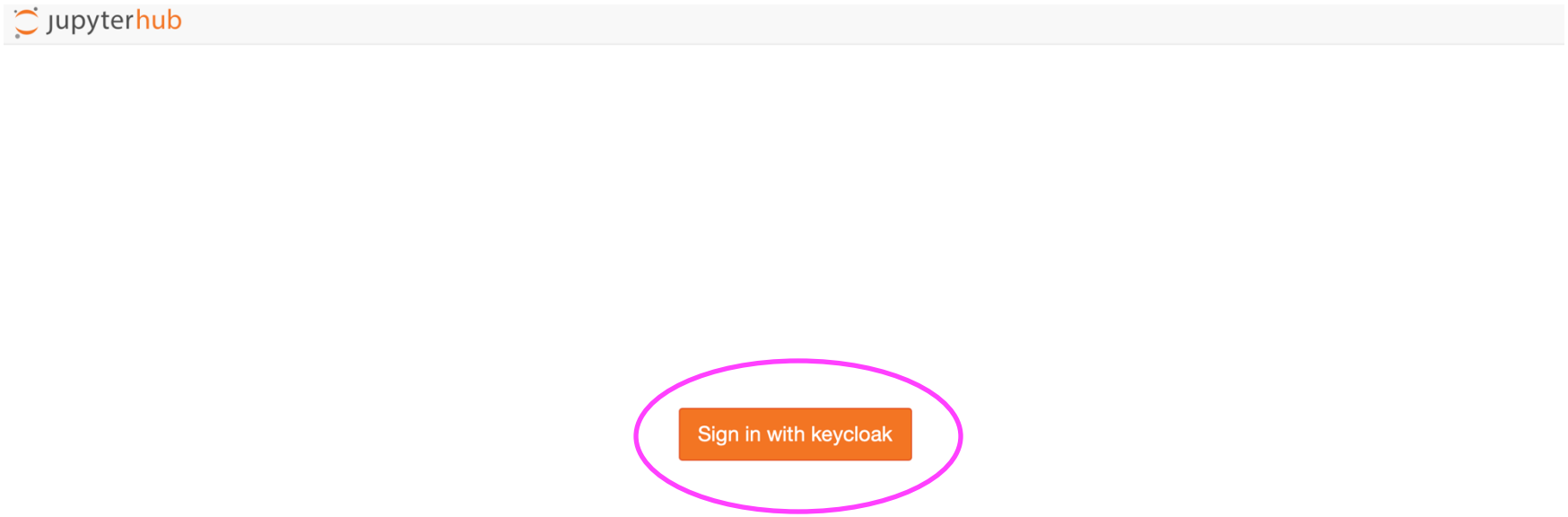
- Download → install → local use

https://omucai.github.io/PyTCT-docs/

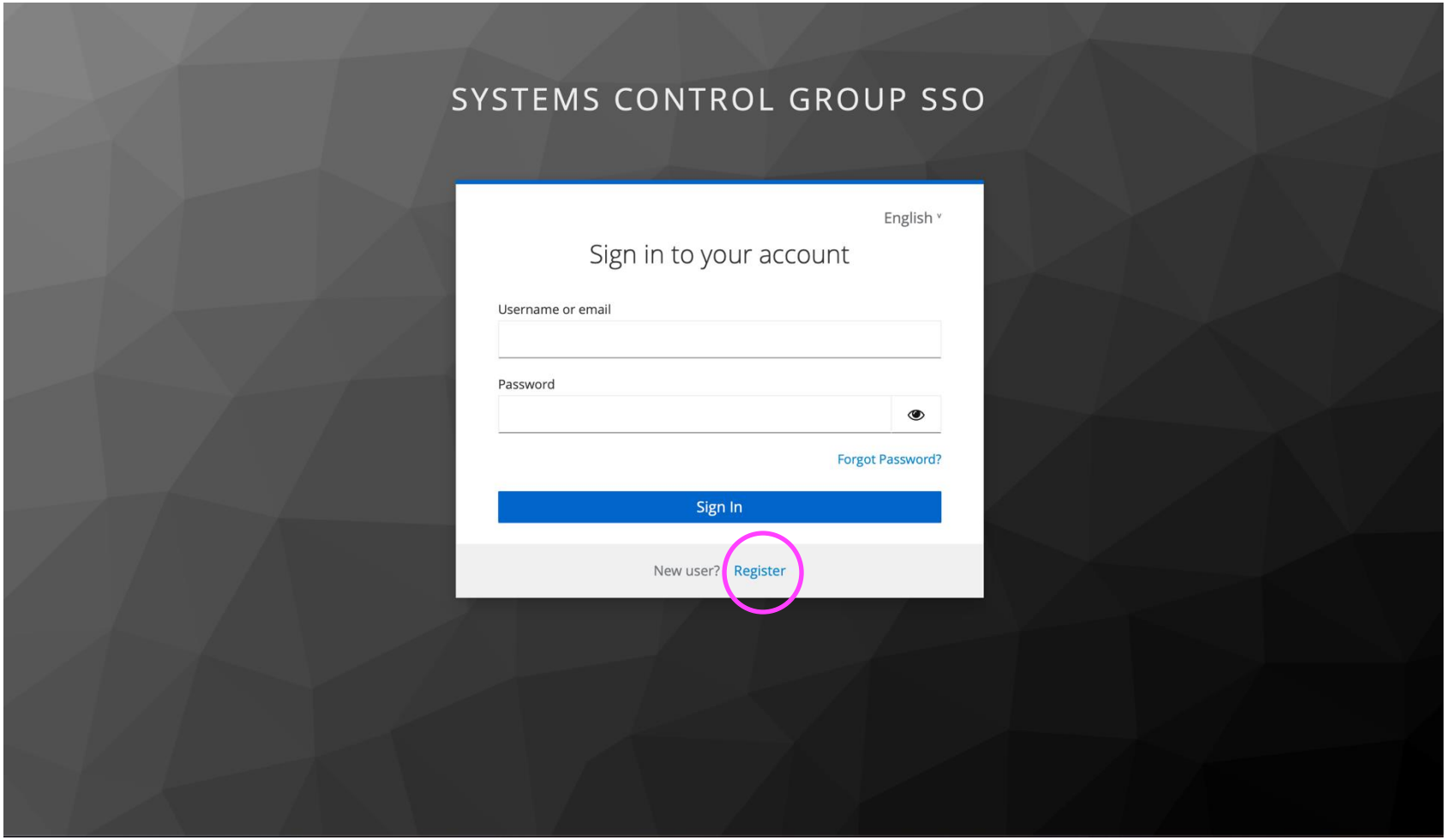
- Download-free, install-free → browser-based online use

https://jupyter.caikai.org

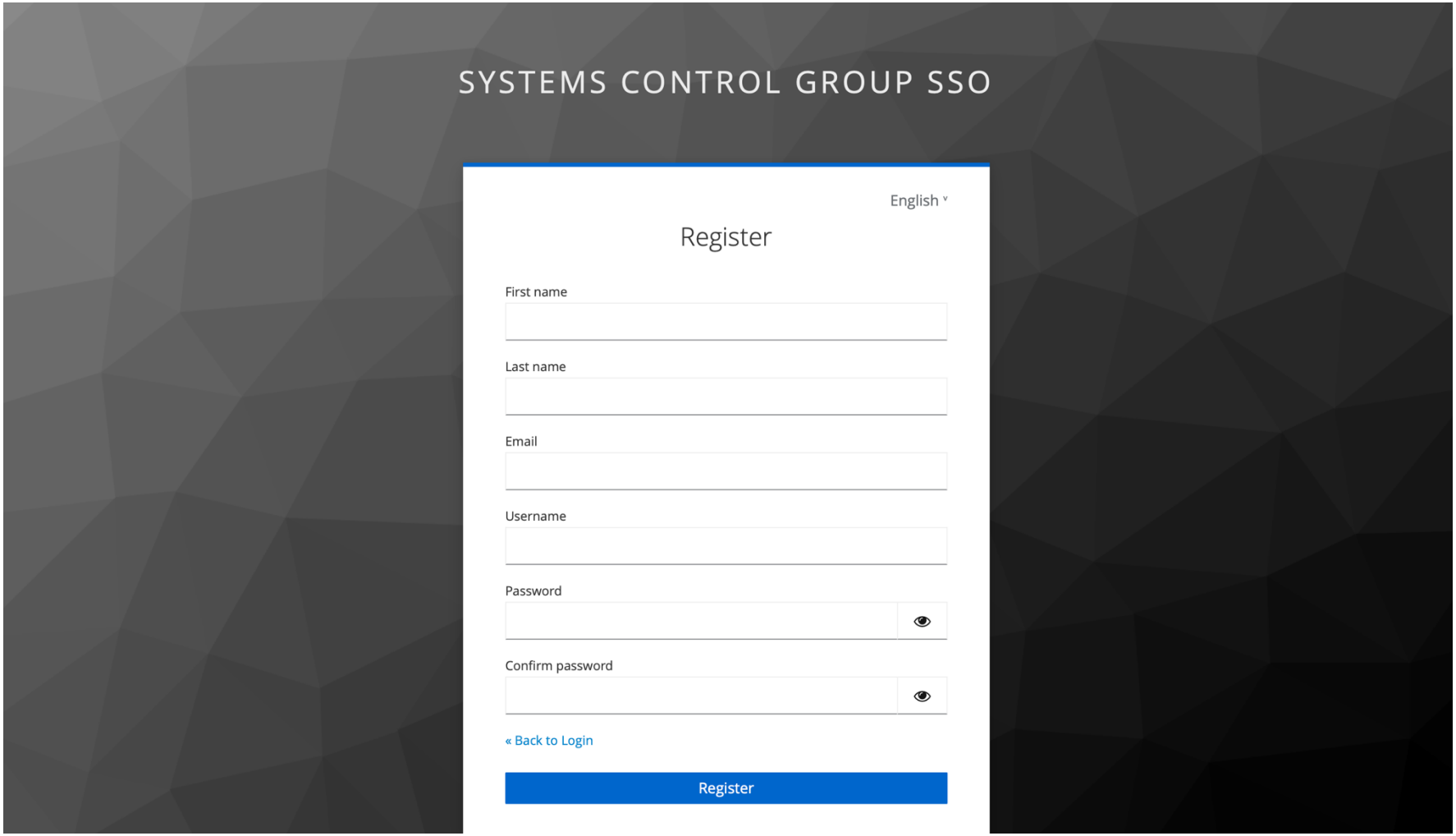
PyTCT



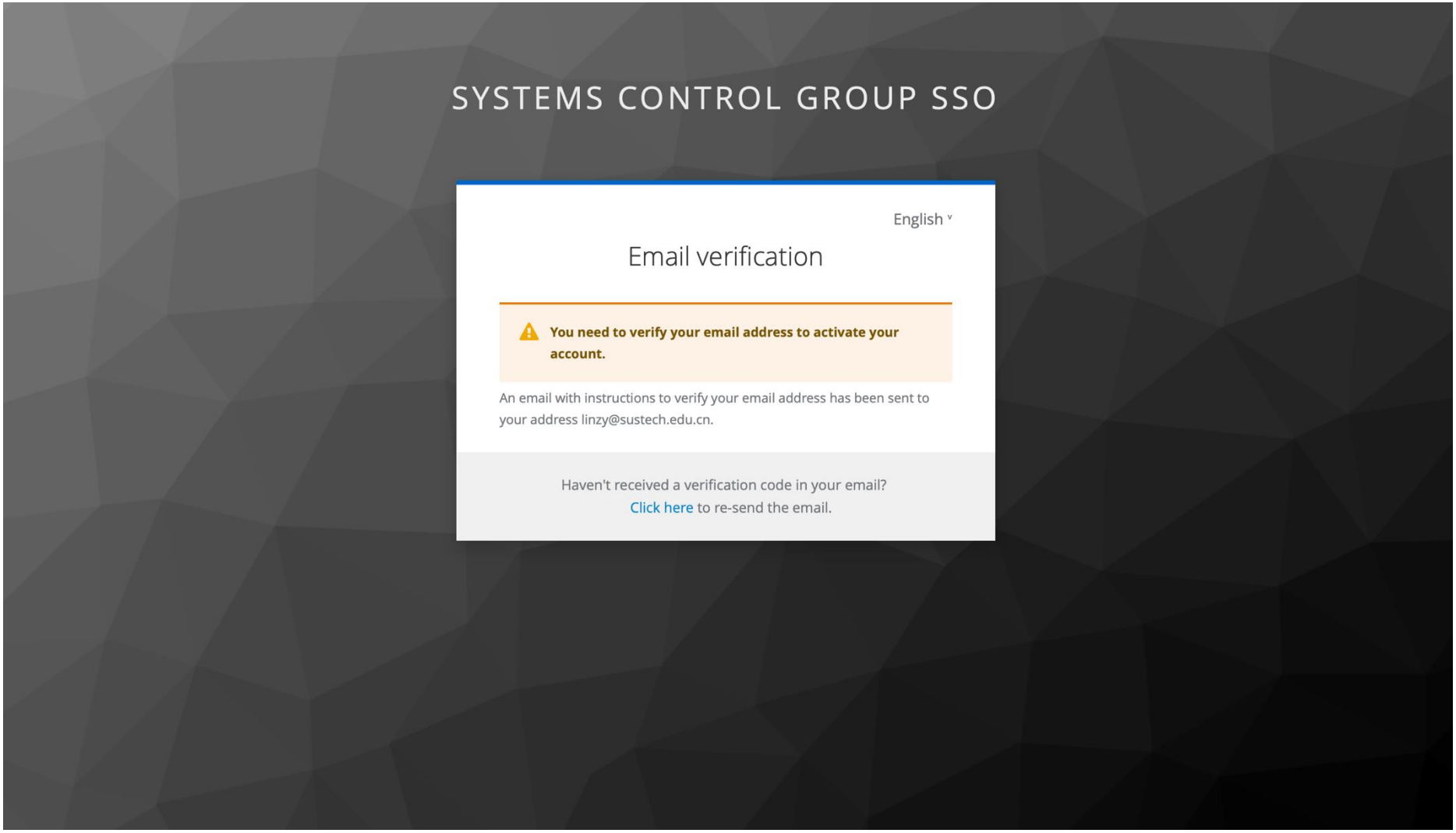
PyTCT


SYSTEMS CONTROL GROUP SSO
English
Sign in to your account
Username or email
Password
Forgot Password?
Sign In
New user?
Register

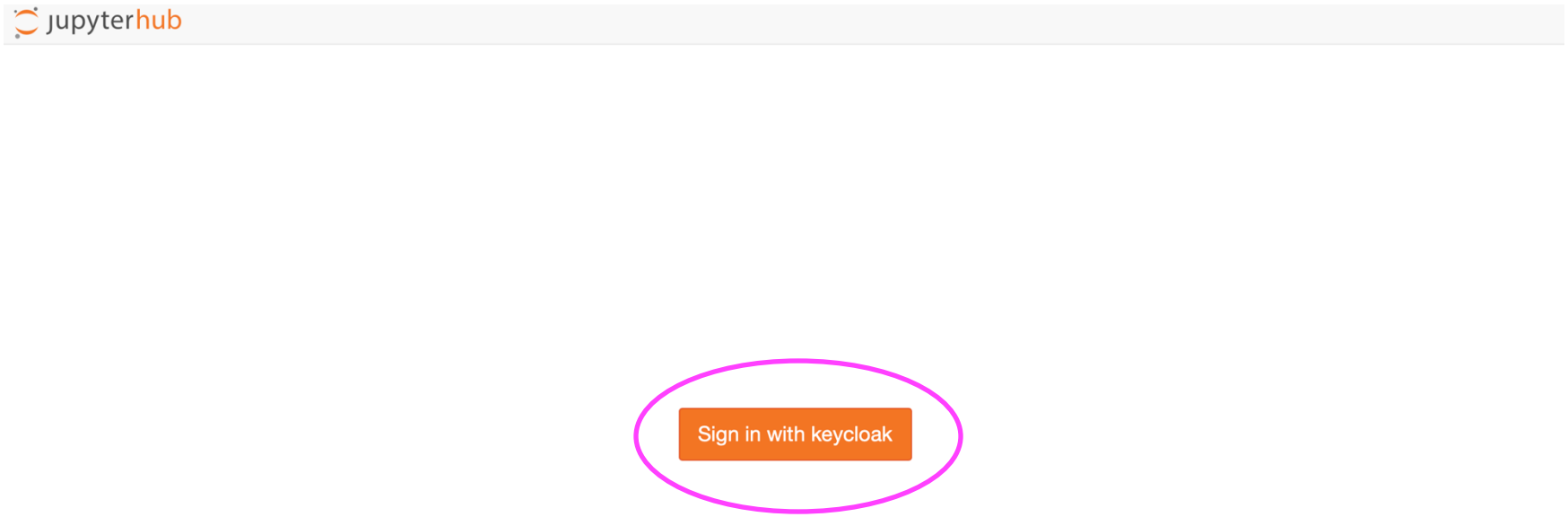
PyTCT


SYSTEMS CONTROL GROUP SSO
English
Register
First name
Last name
Email
Username
Password
Confirm password
« Back to Login
Register

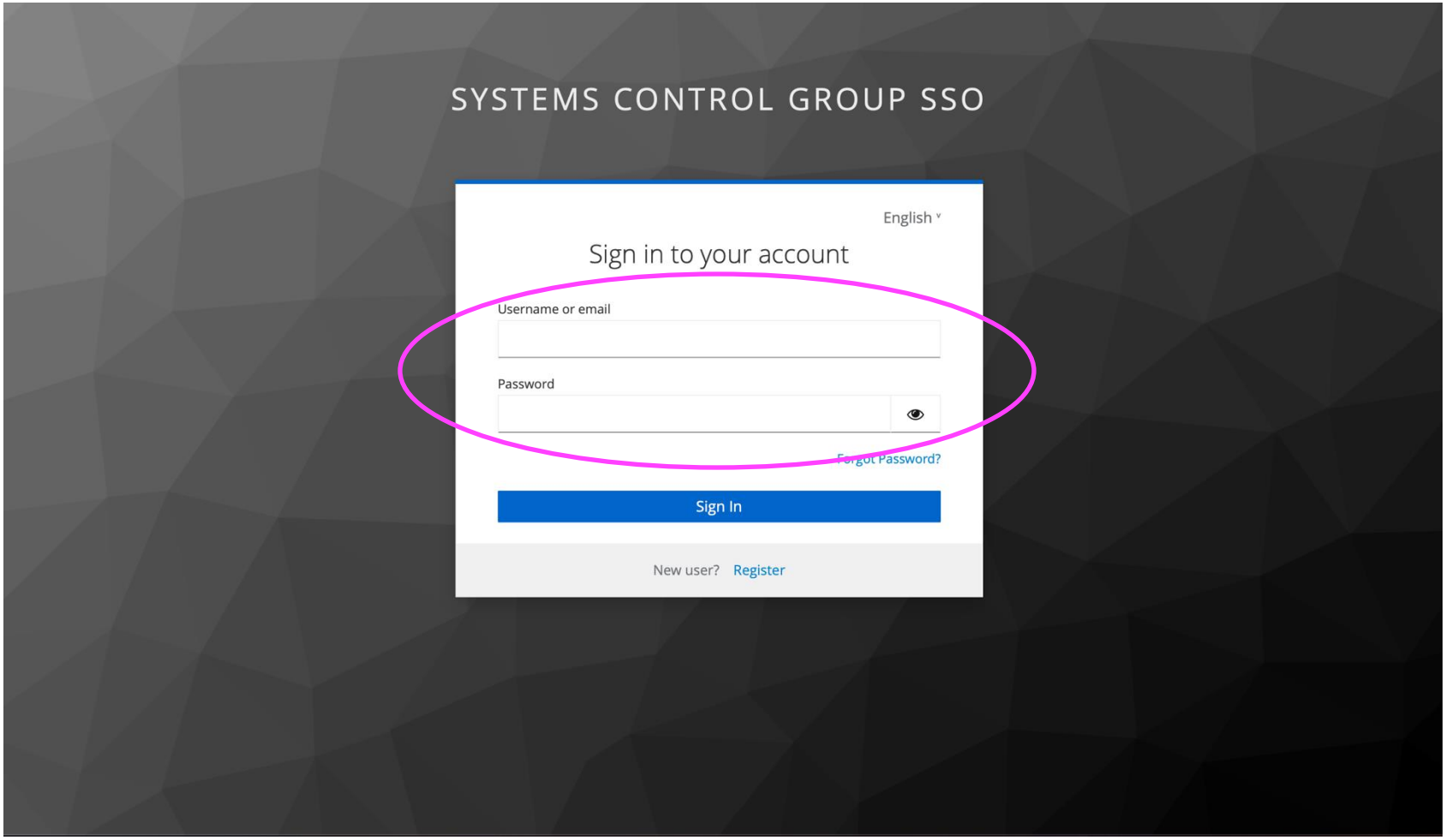
PyTCT


SYSTEMS CONTROL GROUP SSO
English
Email verification
You need to verify your email address to activate your account.
An email with instructions to verify your email address has been sent to your address linzy@sustech.edu.cn.
Haven't received a verification code in your email?
Click here to re-send the email.

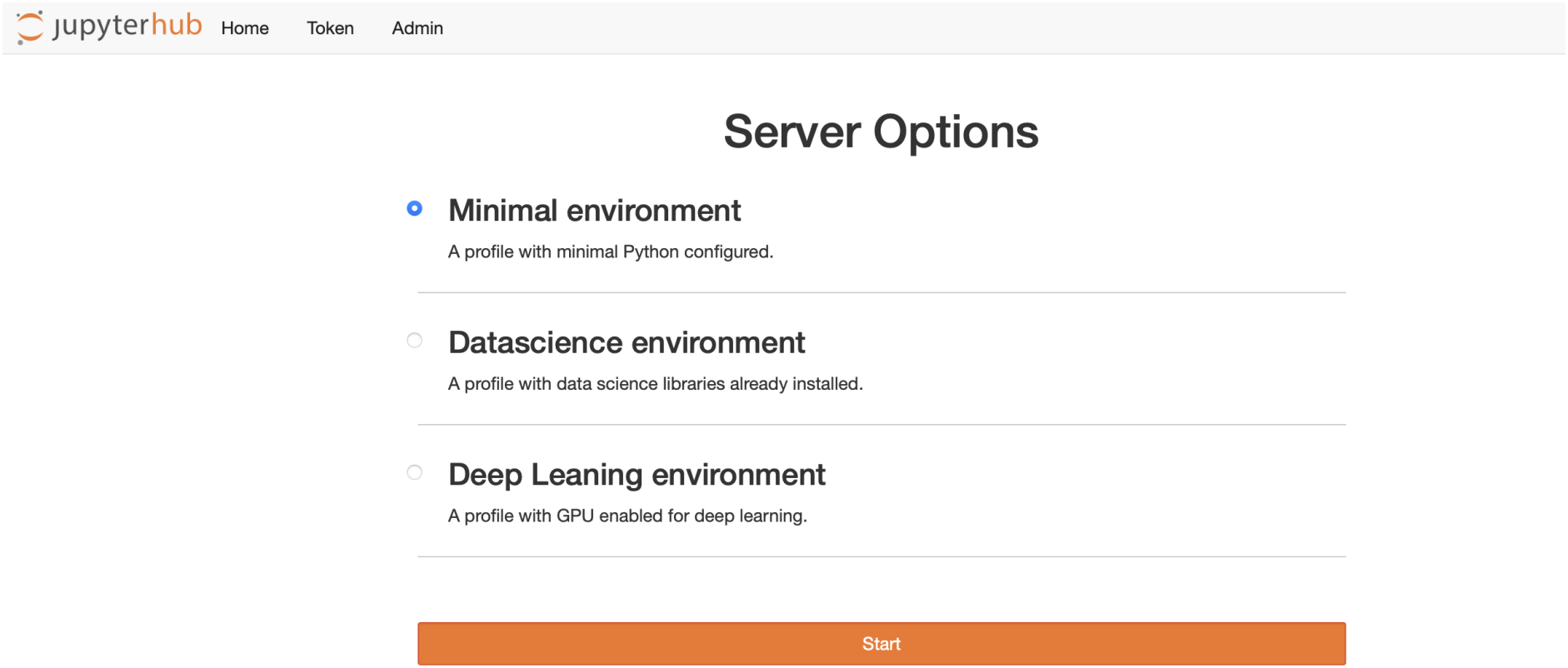
PyTCT



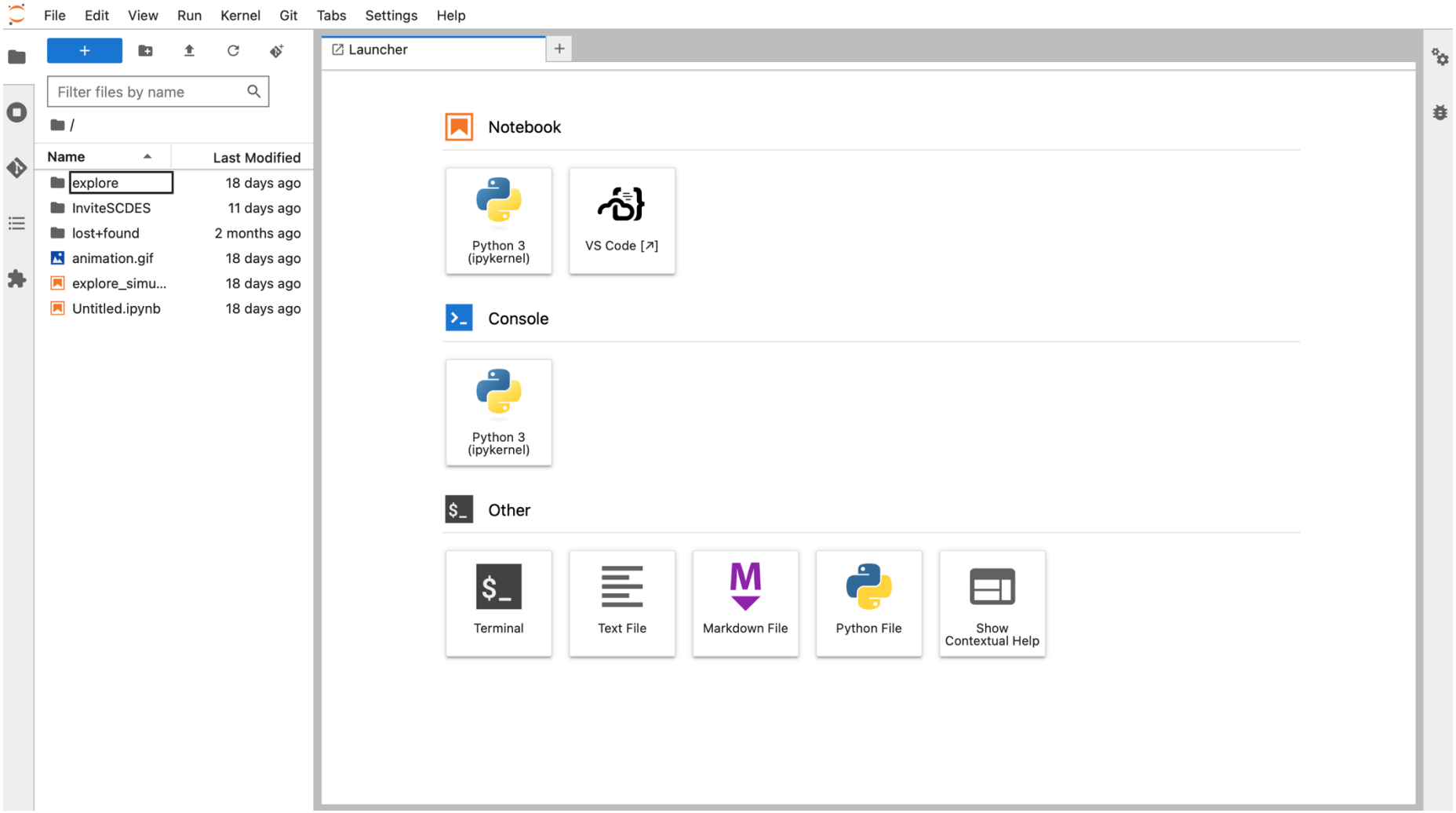
PyTCT


SYSTEMS CONTROL GROUP SSO
English
Sign in to your account
Username or email
Password
Forgot Password?
Sign In
New user? Register

PyTCT


jupyterhub
Home
Token
Admin
Server Options
Minimal environment
A profile with minimal Python configured.
Datascience environment
A profile with data science libraries already installed.
Deep Leaning environment
A profile with GPU enabled for deep learning.
Start

PyTCT


File
Edit
View
Run
Kernel
Git
Tabs
Settings
Help
Launcher
Filter files by name
/
Name
Last Modified
explore
18 days ago
InviteSCDES
11 days ago
lost+found
2 months ago
animation.gif
18 days ago
explore_simu...
18 days ago
Untitled.ipynb
18 days ago
Notebook
Python 3 (ipykernel)
VS Code [↗]
Console
Python 3 (ipykernel)
Other
Terminal
Text File
Markdown File
Python File
Show Contextual Help

PyTCT import and initialization

```
1 import pytct #import pytct package
2
3 pytct.init('xxx') #create a working folder
```

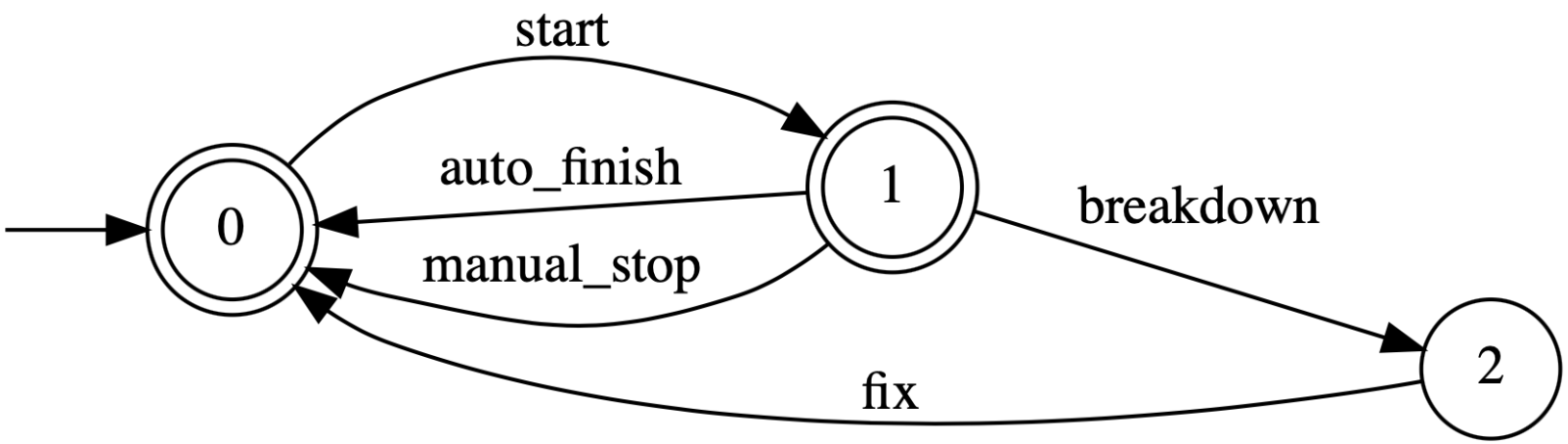
```
1 pytct.init('xxx', overwrite=True)
2 #allow overwriting a created working folder
```

PyTCT create

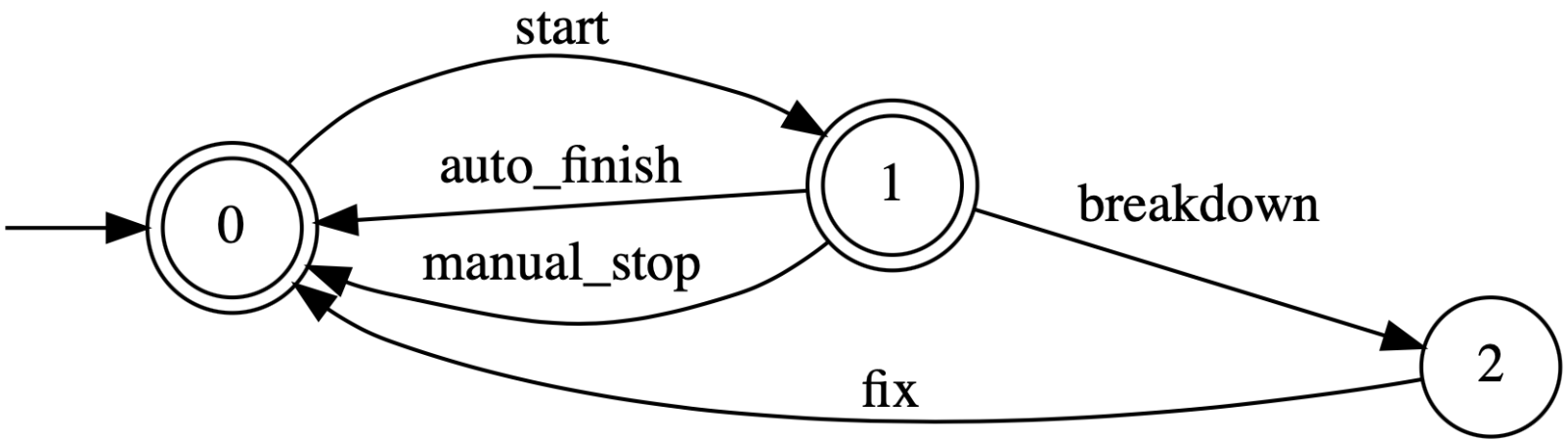
```
statenum=3 #number of states
#states are sequentially labeled 0,1,...,statenum
#initial state is labeled 0

trans=[(0,'start',1),
       (1,'auto_finish',0),
       (1,'manual_stop',0),
       (1,'breakdown',2),
       (2,'fix',0)] # set of transitions
#each triple is (exit state, event label, entering state)

marker = [0,1] #set of marker states

pytct.create('PRINTER', statenum, trans, marker)
```

PyTCT display_automaton

```
1 pytct.display_automaton('PRINTER')
2 #plot PRINTER.DES
```

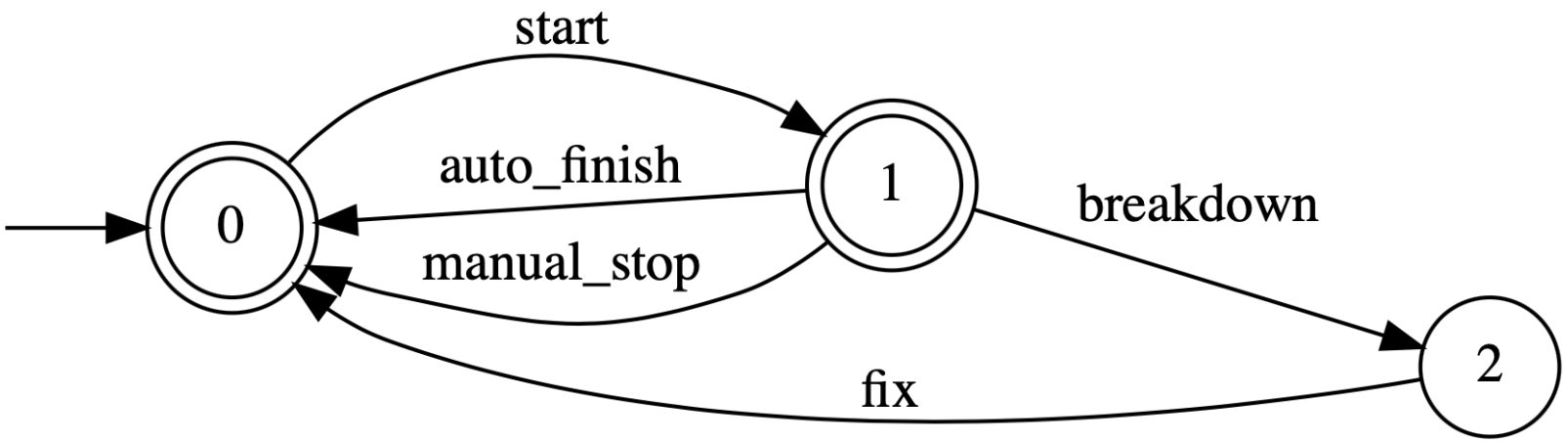


PyTCT information extraction




```
1 pytct.statenum('A') #state number of A.DES
2
3 pytct.events('A') #events of A.DES
4
5 pytct.trans('A') #state transitions of A.DES
6
7 pytct.marker('A') #marker states of A.DES
```

PyTCT information extraction




```
pytct.statenum('PRINTER')
 3
```

```
pytct.events('PRINTER')
['manual_stop', 'breakdown', 'auto_finish', 'fix', 'start']
```

```
pytct.trans('PRINTER')
[(0, 'start', 1),
 (1, 'auto_finish', 0),
 (1, 'manual_stop', 0),
 (1, 'breakdown', 2),
 (2, 'fix', 0)]
```

```
pytct.marker('PRINTER')
 [0, 1]
```

State transition as function

State transition function

Let $\mathbf{G} = (Q, \Sigma, \delta, q_0, Q_m)$ be an automaton.

δ : finite transition set

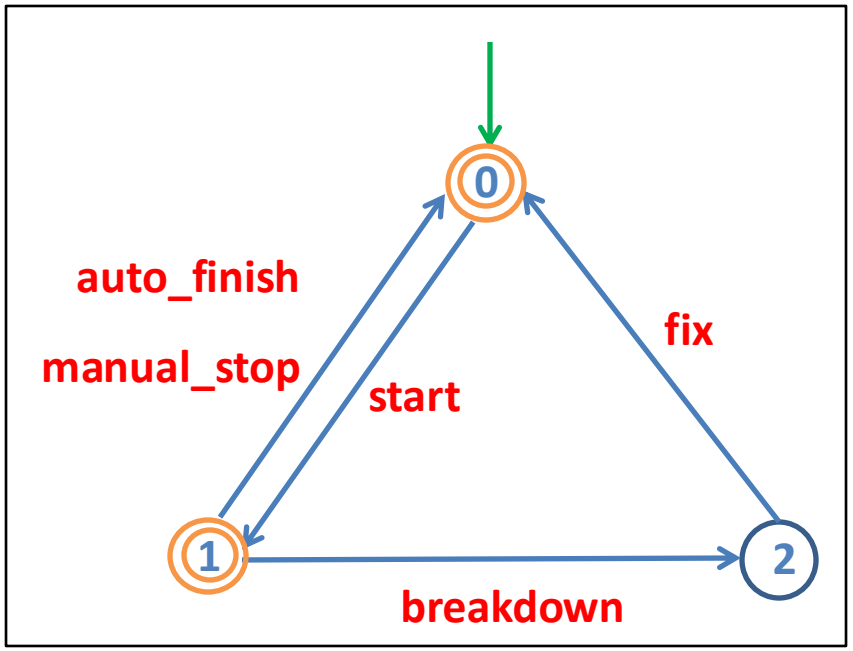
$$\delta \subseteq \{(q, \sigma, q') \mid q, q' \in Q, \sigma \in \Sigma\}$$

is viewed as a function $\delta : Q \times \Sigma \to Q$

- domain: $Q \times \Sigma$
- δ is *partial*: not all pairs (q, σ) are defined
- write $\delta(q, \sigma)!$ to mean σ is defined at q, and $\neg\delta(q, \sigma)!$ otherwise

A printer

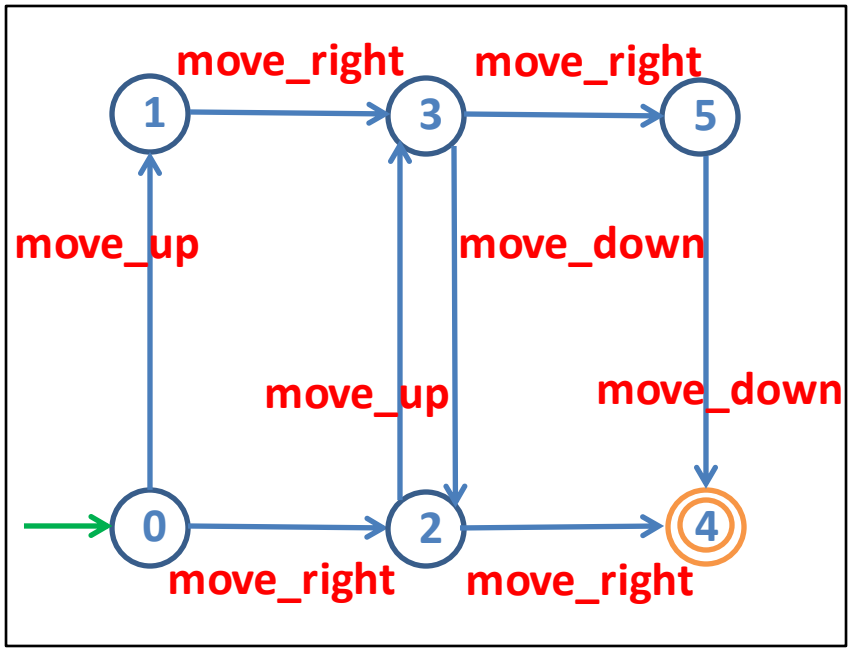
$$\mathbf{PRINTER} = (Q, \Sigma, \delta, q_0, Q_m)$$



$$\delta : Q \times \Sigma \rightarrow Q$$

A warehouse robot

$$\mathbf{ROBOT} = (Q, \Sigma, \delta, q_0, Q_m)$$



$$\delta : Q \times \Sigma \rightarrow Q$$

Multi-step transition

Function δ can be 'composed' to describe multi-step transition

e.g. $\delta(q_1, \sigma_1)!$ & $\delta(q_1, \sigma_1) = q_2$

$\delta(q_2, \sigma_2)!$ & $\delta(q_2, \sigma_2) = q_3$

$\delta(q_3, \sigma_3)!$ & $\delta(q_3, \sigma_3) = q_4$

$\Rightarrow$ 3-step transition:

$$\delta(\delta(\delta(q_1, \sigma_1), \sigma_2), \sigma_3) = q_4$$

$$\hat{\delta}(q_1, \sigma_1\sigma_2\sigma_3) = q_4 \text{ (compactly)}$$

a string on Σ

Strings

A (finite) string on Σ is a finite sequence of events from Σ:

$$s = \sigma_1 \sigma_2 \cdots \sigma_k, \quad \sigma_i \in \Sigma, \ i = 1, ..., k$$

Denote by Σ^+ the set of all strings on Σ

Strings

Let ϵ be the empty string ($\epsilon \notin \Sigma^+$)

Write $\Sigma^* = \{\epsilon\} \dot{\cup} \Sigma^+$ ("Kleene star")

State transition function

Let's extend δ to $\hat{\delta} : Q \times \Sigma^* \to Q$

$$\hat{\delta}(q, \epsilon) = q, \quad q \in Q$$

$$\hat{\delta}(q, \sigma) = \delta(q, \sigma), \quad q \in Q, \sigma \in \Sigma$$

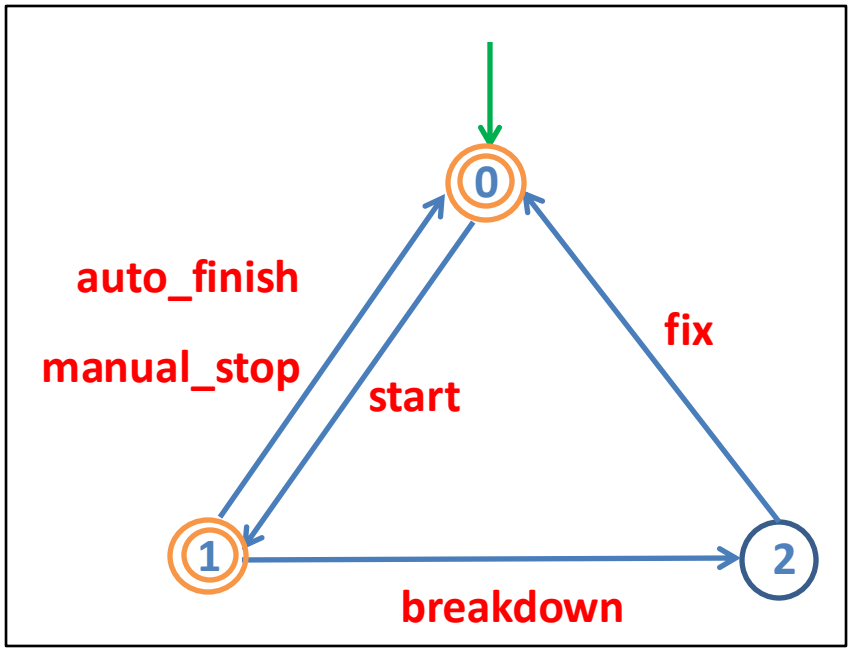
$$\hat{\delta}(q, s\sigma) = \delta(\hat{\delta}(q, s), \sigma), \quad q \in Q, s \in \Sigma^*, \sigma \in \Sigma$$

Henceforth drop $\hat{}$ and just write δ.
Also write $\delta(q, s)!$ to mean $s \in \Sigma^*$ is defined at $q \in Q$.

Note: $\delta(q, \epsilon)!$ for every state $q \in Q$.

A printer

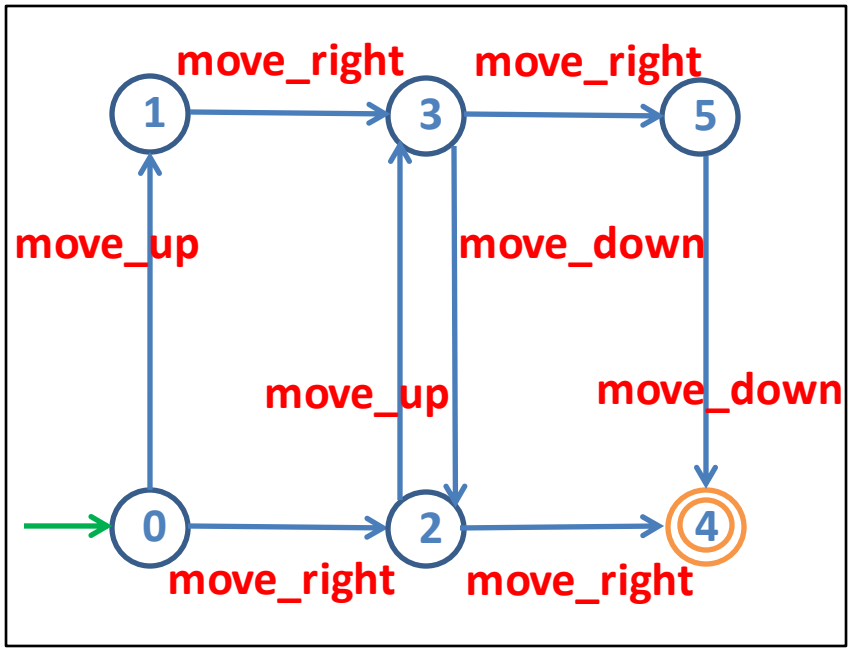
$$\mathbf{PRINTER} = (Q, \Sigma, \delta, q_0, Q_m)$$



$$\delta : Q \times \Sigma^* \rightarrow Q$$

A warehouse robot

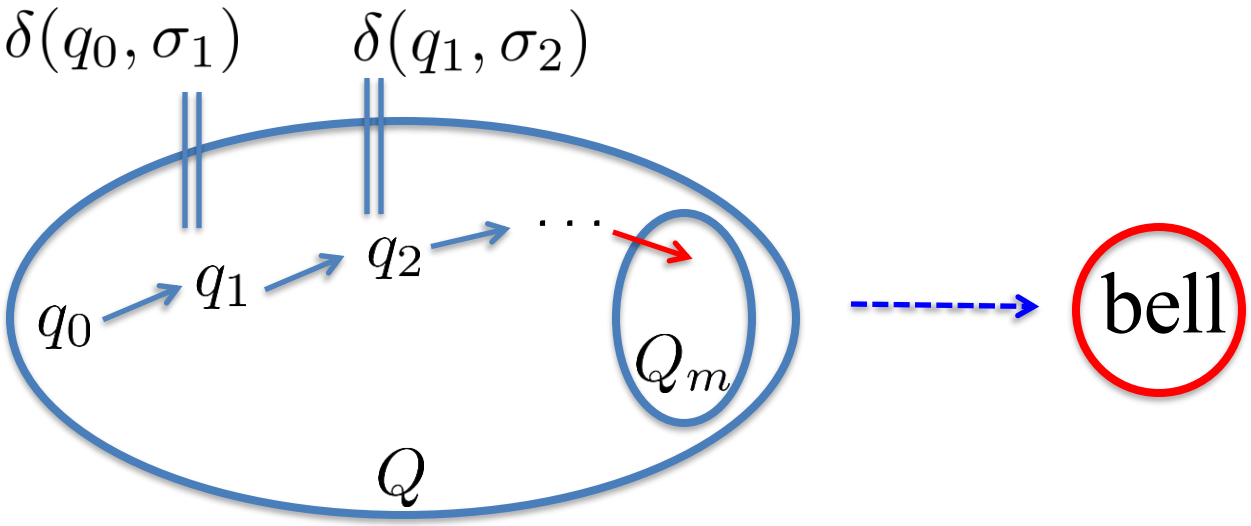
$$\mathbf{ROBOT} = (Q, \Sigma, \delta, q_0, Q_m)$$



$$\delta : Q \times \Sigma^* \rightarrow Q$$

Automaton as dynamic system

Automaton $\mathbf{G} = (Q, \Sigma, \delta, q_0, Q_m)$ is a dynamic system



Internal state transitions

Beeps when a transition enters a marker state

Automaton as dynamic system

Automaton $\mathbf{G} = (Q, \Sigma, \delta, q_0, Q_m)$ is a dynamic system

A string starting from q_0 is a *trajectory* of $\mathbf{G}$

Set of all trajectories of $\mathbf{G}$:

$$L(\mathbf{G}) := \{s \in \Sigma^* \mid \delta(q_0, s)!\} \subseteq \Sigma^*$$

a language on Σ

$L(\mathbf{G})$ is closed behavior of $\mathbf{G}$

Automaton as dynamic system

Automaton $\mathbf{G} = (Q, \Sigma, \delta, q_0, Q_m)$ is a dynamic system

A string starting from q_0 is a *trajectory* of $\mathbf{G}$

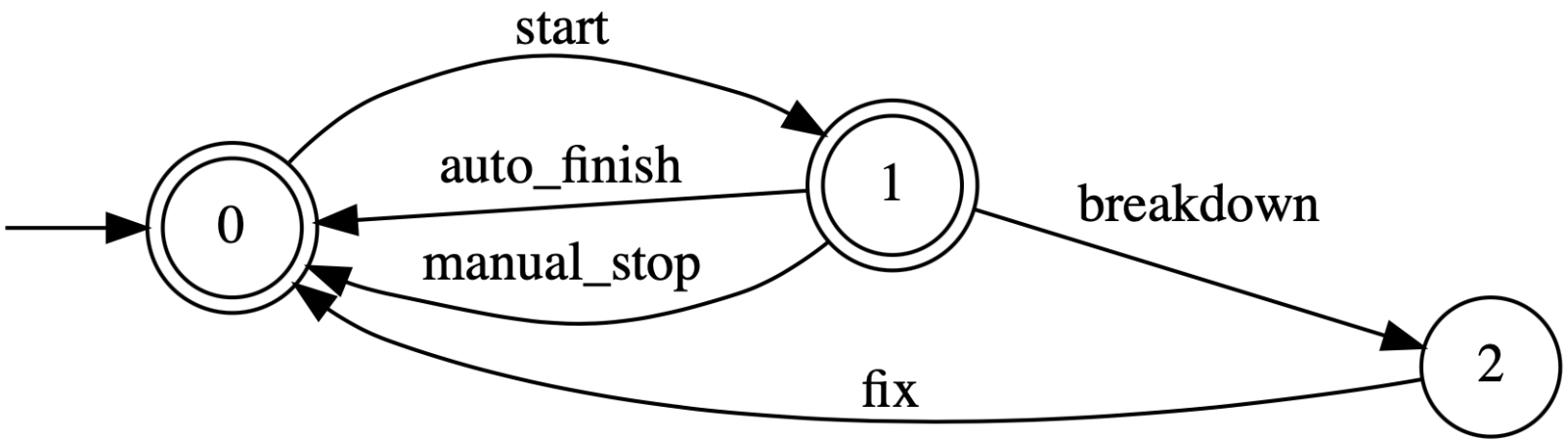
Set of 'good' trajectories of $\mathbf{G}$:

$$L_m(\mathbf{G}) := \{s \in L(\mathbf{G}) \mid \delta(q_0, s) \in Q_m\} \subseteq \Sigma^*$$

also language on Σ

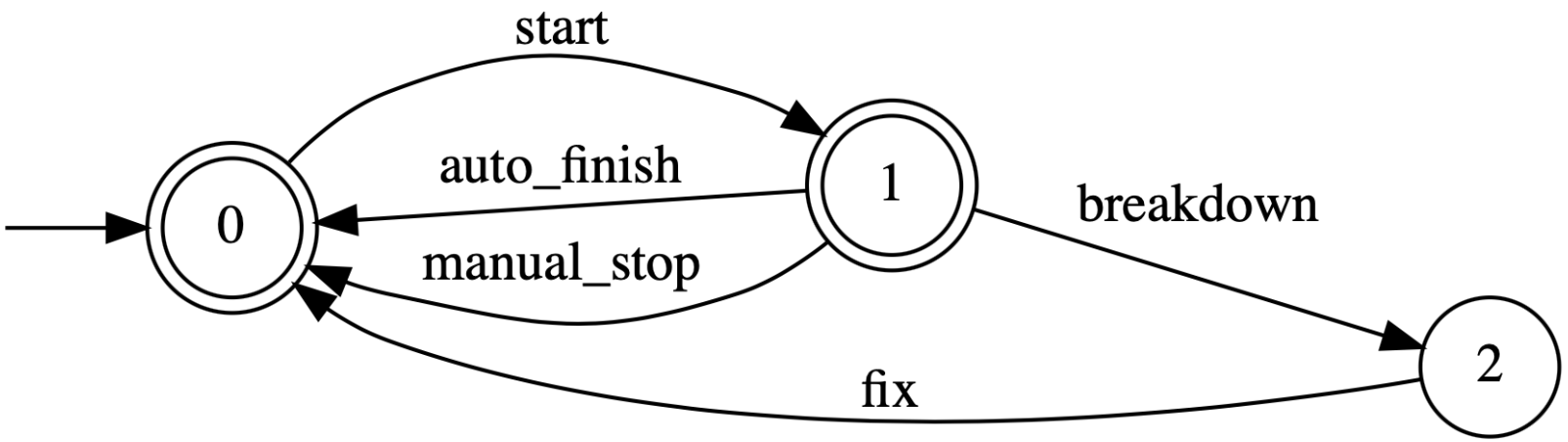
$L_m(\mathbf{G})$ is marked behavior of $\mathbf{G}$

PyTCT simulate




```
pytct.simulate_automaton('PRINTER',['start','breakdown','
    fix'])
#simulate PRINTER.DES with a string
```

PyTCT sample




```
pytct.sample_automaton('PRINTER',5])
#sample PRINTER.DES with a string of length 5
```

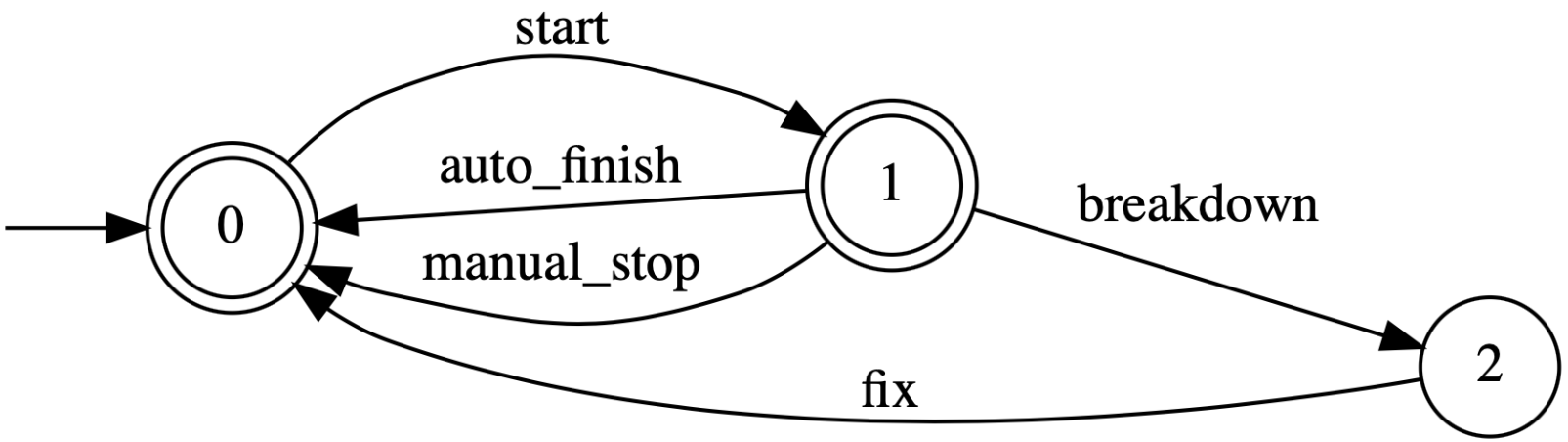
```
pytct.sample_automaton('PRINTER',5)
```

```
[0, 1, 2, 0, 1, 0]
```

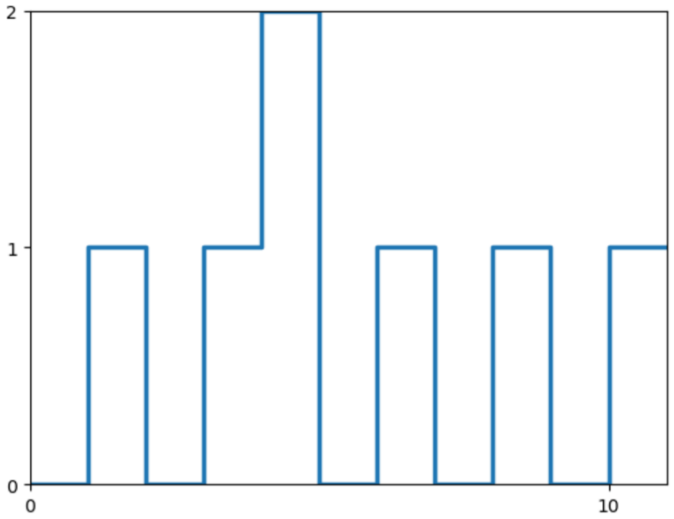
```
pytct.sample_automaton('PRINTER',5)
```

```
[0, 1, 0, 1, 0, 1]
```

PyTCT sample




```
import matplotlib.pyplot as plt
import numpy as np

# create x, y data
y = [0] + pytct.sample_automaton('PRINTER', 10, strict=False)
# strict = False: ignore error if length of sampled string
# is shorter than specified due to blocking
x = np.arange(len(y))

# plot
fig, ax = plt.subplots()
ax.step(x, y, linewidth=2.5)
ax.set(xlim=(0, max(x)), xticks=np.arange(0, max(x),10),
       ylim=(0, max(y)), yticks=np.arange(0, max(y)+1))

plt.show()
```



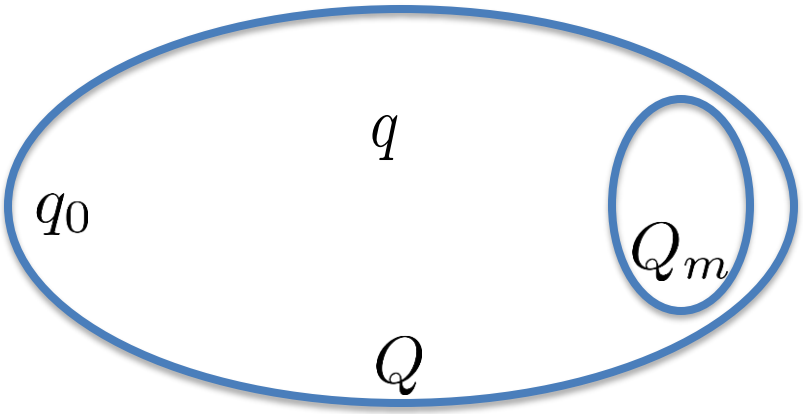
("Datascience" environment)

Properties of an automaton

Reachable

Let $\mathbf{G} = (Q, \Sigma, \delta, q_0, Q_m)$ be an automaton.

A state $q \in Q$ is reachable if $(\exists s \in \Sigma^*)\delta(q_0, s) = q$.

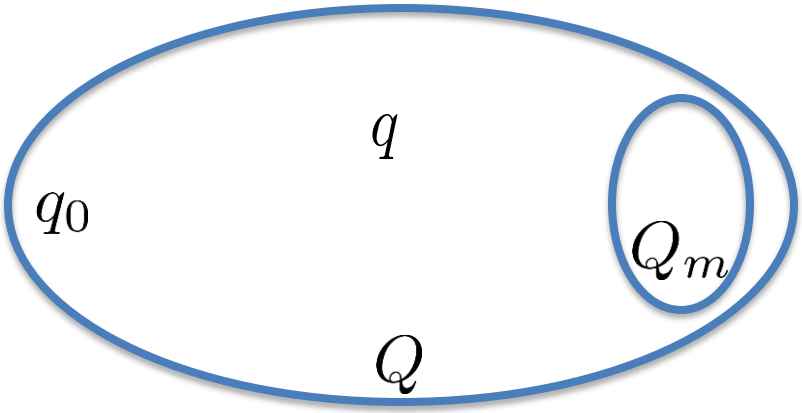


$\mathbf{G}$ is reachable if every state $q \in Q$ is reachable.

Coreachable

Let $\mathbf{G} = (Q, \Sigma, \delta, q_0, Q_m)$ be an automaton.

A state $q \in Q$ is coreachable if $(\exists s \in \Sigma^*)\delta(q, s) \in Q_m$.



$\mathbf{G}$ is coreachable if every state $q \in Q$ is coreachable.

Trim

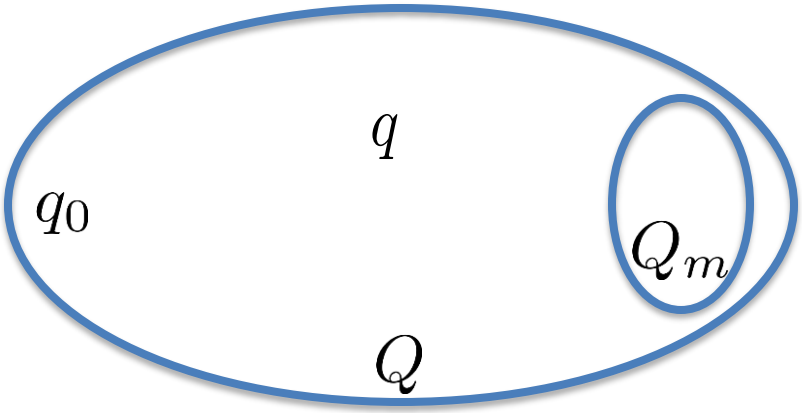
Let $\mathbf{G} = (Q, \Sigma, \delta, q_0, Q_m)$ be an automaton.

$\mathbf{G}$ is trim if it is both reachable and coreachable.

Nonblocking

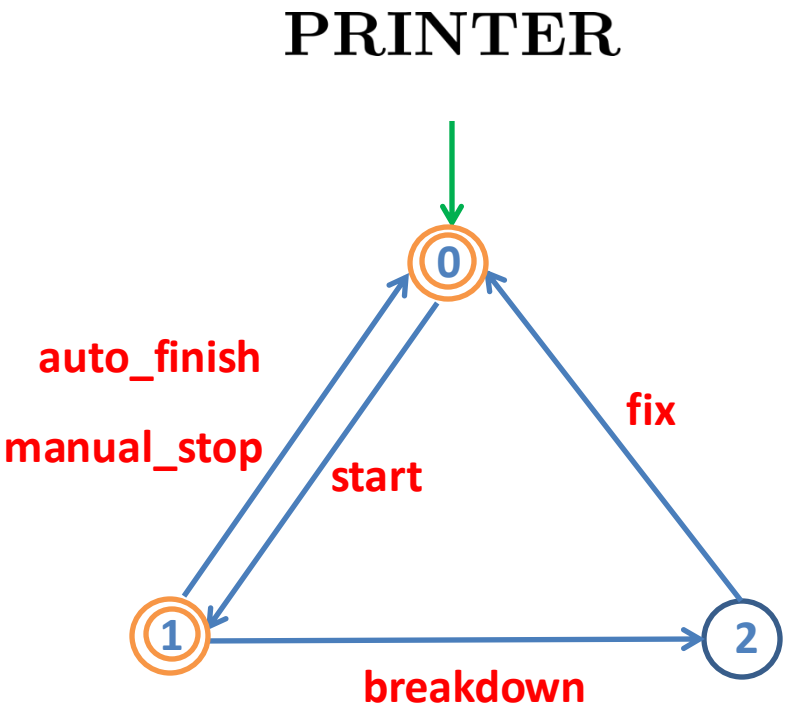
Let $\mathbf{G} = (Q, \Sigma, \delta, q_0, Q_m)$ be an automaton.

$\mathbf{G}$ is nonblocking if $(\forall q \in Q) q$ is reachable $\Rightarrow q$ is coreachable.



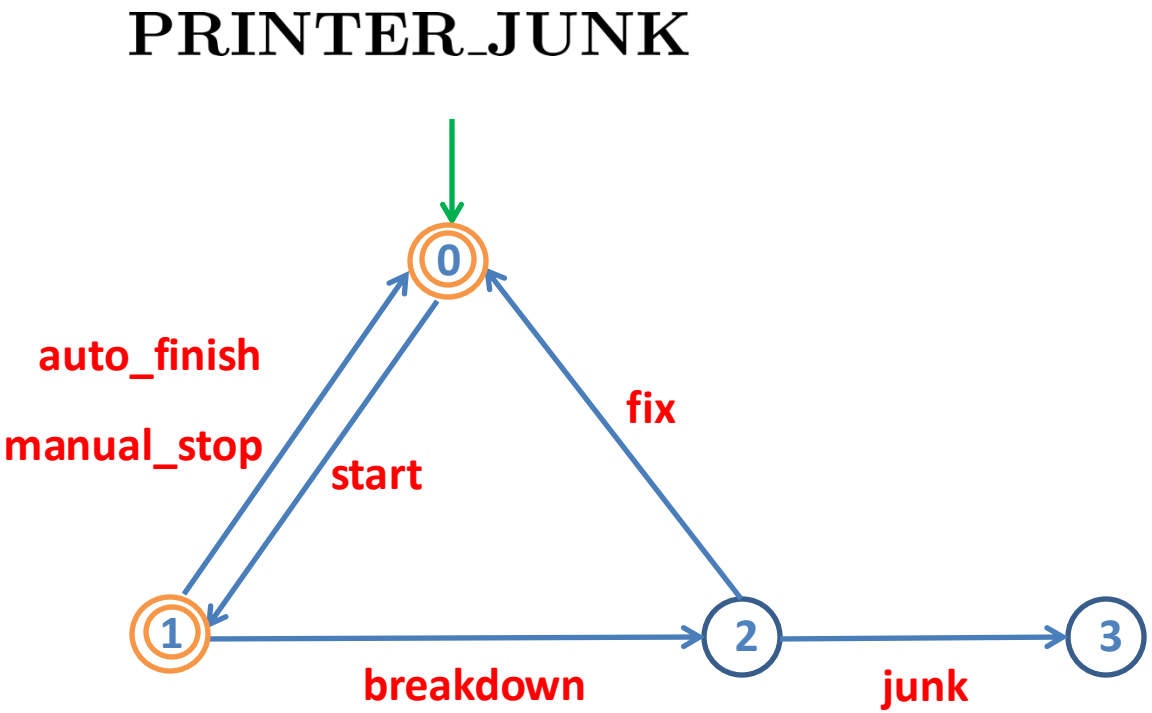
If $\mathbf{G}$ is trim, then $\mathbf{G}$ is nonblocking;
but reverse is false (why?)

A printer



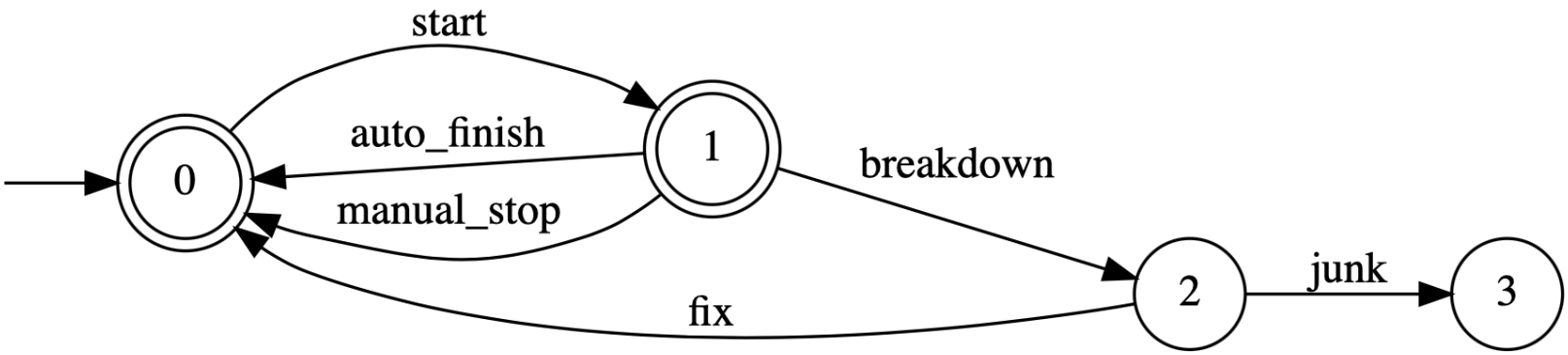
Is **PRINTER** reachable, coreachable, trim, nonblocking?

A printer



Is **PRINTER** reachable, coreachable, trim, nonblocking?

PyTCT reachable




```
pytct.is_reachable('PRINTER_JUNK')
```

```
True
```

```
pytct.is_reachable('PRINTER_JUNK',3)
```

```
True
```

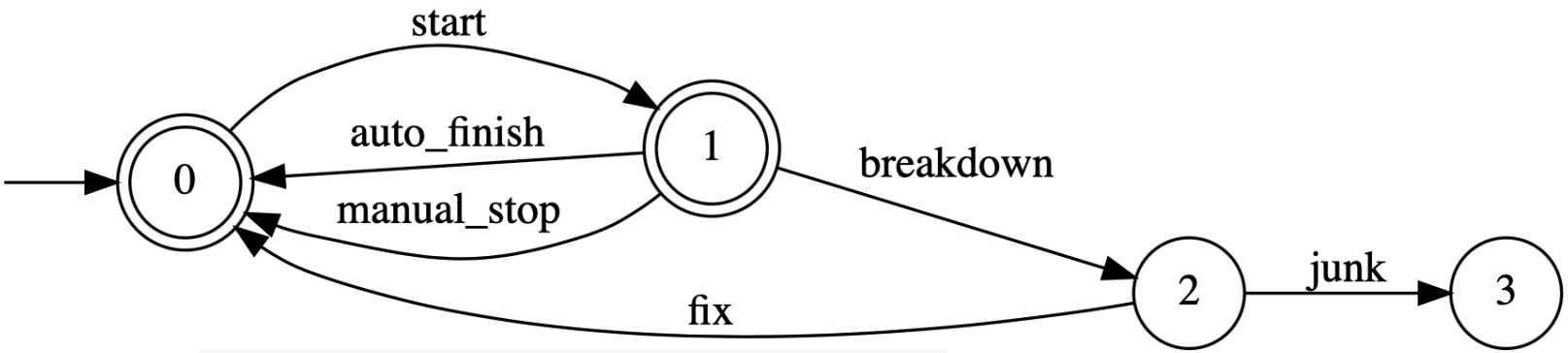
```
# 0(initial state) -> 3
pytct.reachable_string('PRINTER_JUNK', 3)
```

```
['start', 'breakdown', 'junk']
```

```
for i in range(statenum):
    print(f"state {i}: reachable -> {pytct.is_reachable('PRINTER_JUNK', i)}")
```

```
state 0: reachable -> True
state 1: reachable -> True
state 2: reachable -> True
state 3: reachable -> True
```

PyTCT coreachable




```
pytct.is_coreachable('PRINTER_JUNK')
```

```
False
```

```
pytct.is_coreachable('PRINTER_JUNK',3)
```

```
False
```

```
pytct.is_coreachable('PRINTER_JUNK',2)
```

```
True
```

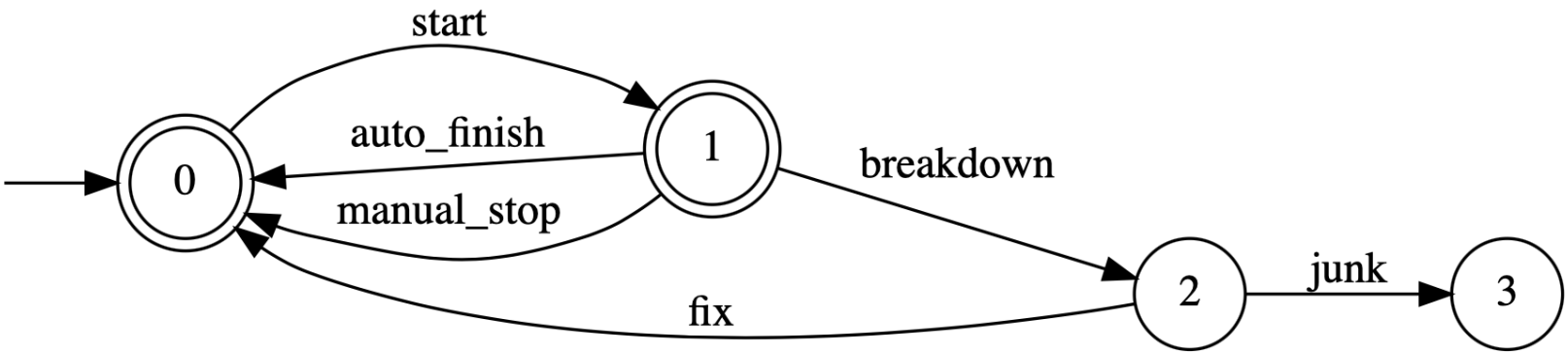
```
pytct.coreachable_string('PRINTER_JUNK', 2)
```

```
['fix']
```

```
for i in range(statenum):
    print(f"state {i}: coreachable -> {pytct.is_coreachable('PRINTER_JUNK', i)}")
```

```
state 0: coreachable -> True
state 1: coreachable -> True
state 2: coreachable -> True
state 3: coreachable -> False
```

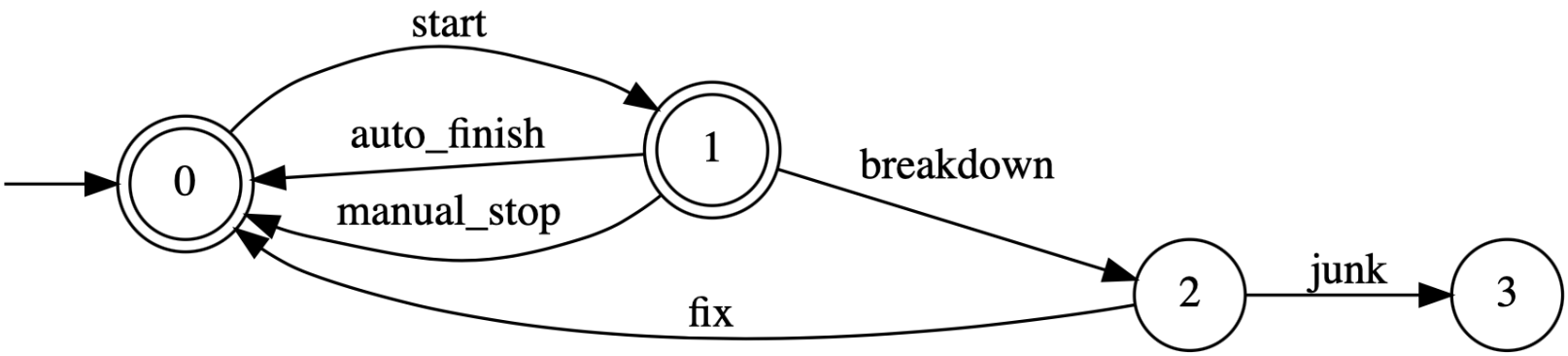
PyTCT shortest path




```
pytct.shortest_string('PRINTER_JUNK', 1, 3)
```

```
['breakdown', 'junk']
```

PyTCT nonblocking




```
pytct.is_nonblocking('PRINTER_JUNK')
```

```
False
```

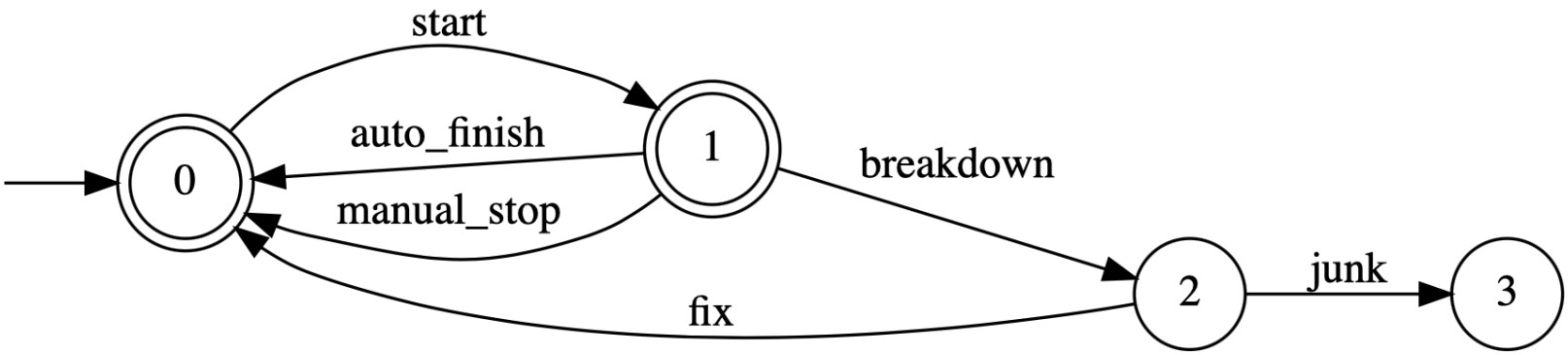
```
pytct.blocking_states('PRINTER_JUNK')
```

```
[3]
```

```
# 0(initial state) -> 3
pytct.reachable_string('PRINTER_JUNK', 3)
```

```
['start', 'breakdown', 'junk']
```

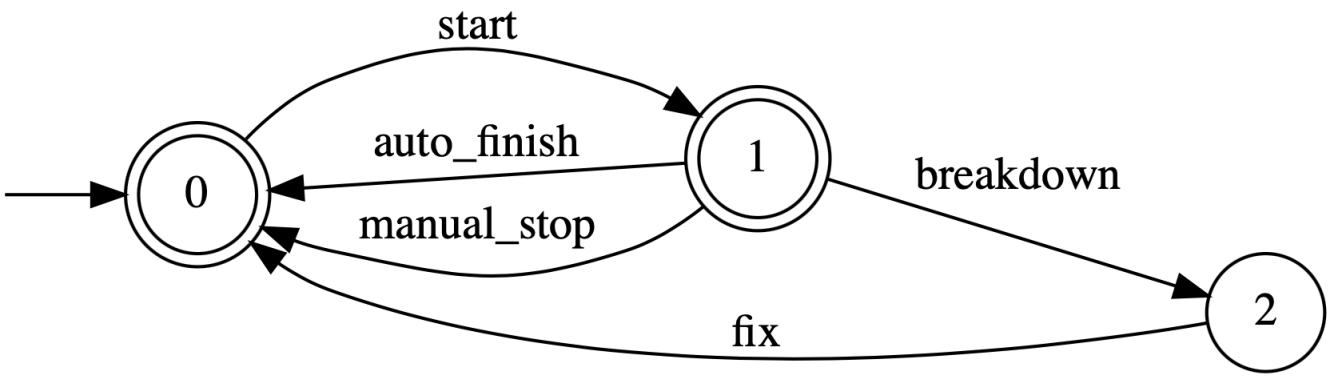
PyTCT trim




```
pytct.is_trim('PRINTER_JUNK')
```

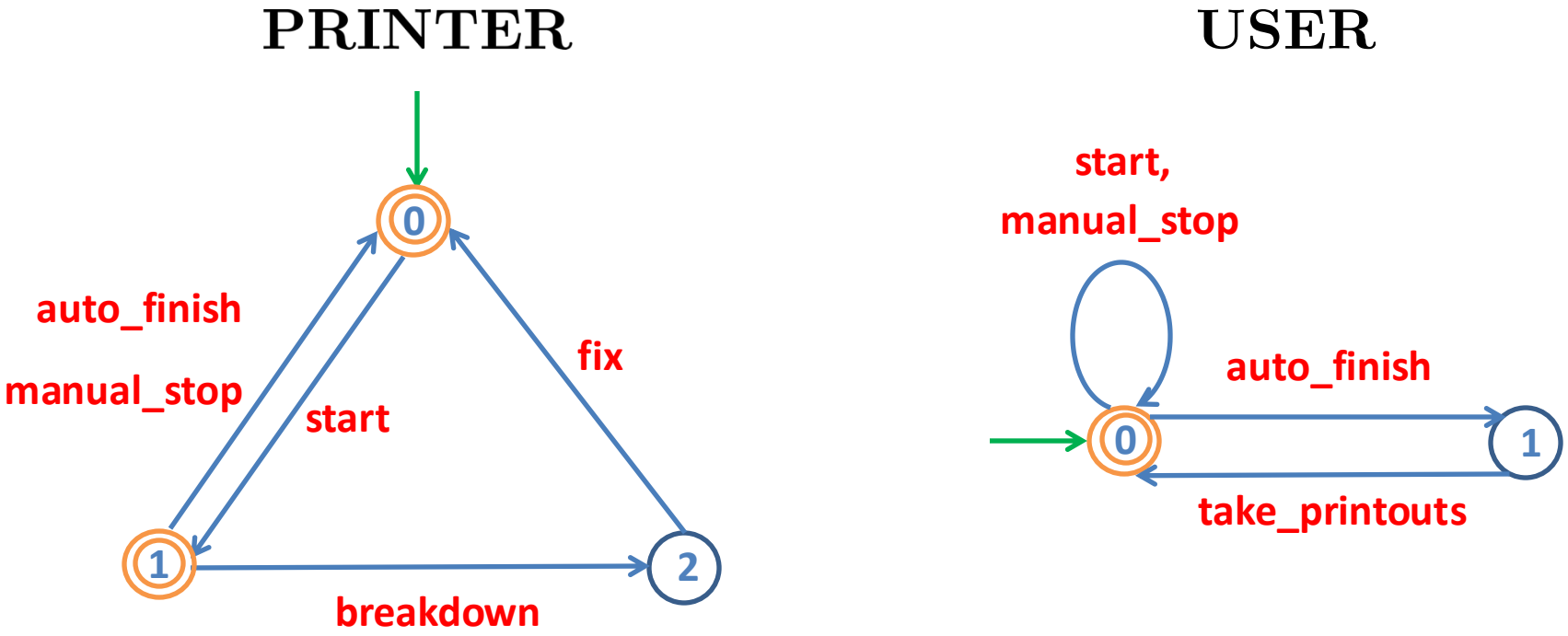
```
False
```

```
pytct.trim('PRINTER_TRIM','PRINTER_JUNK')
pytct.display_automaton('PRINTER_TRIM')
```

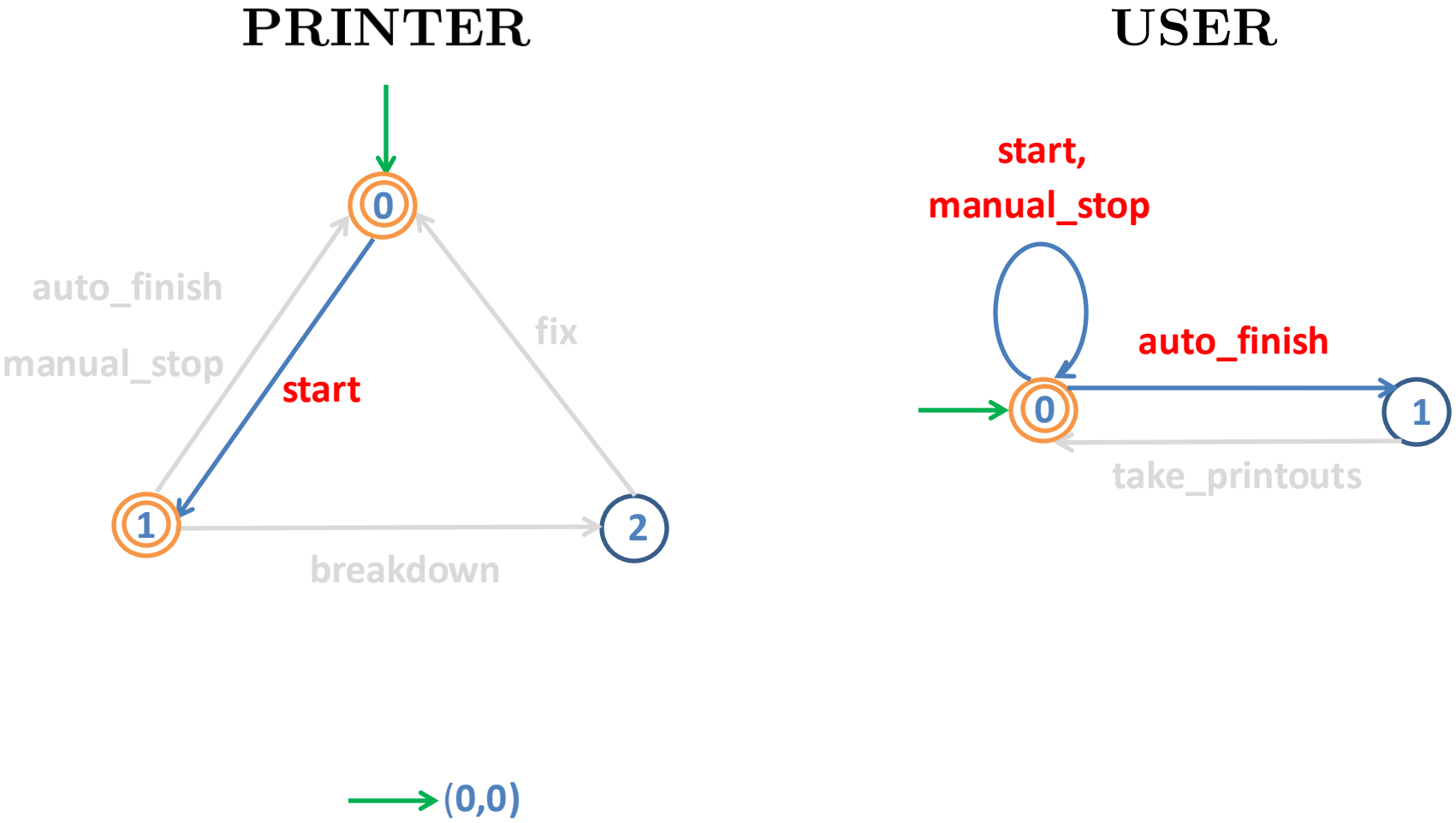


Synchronous product of automata

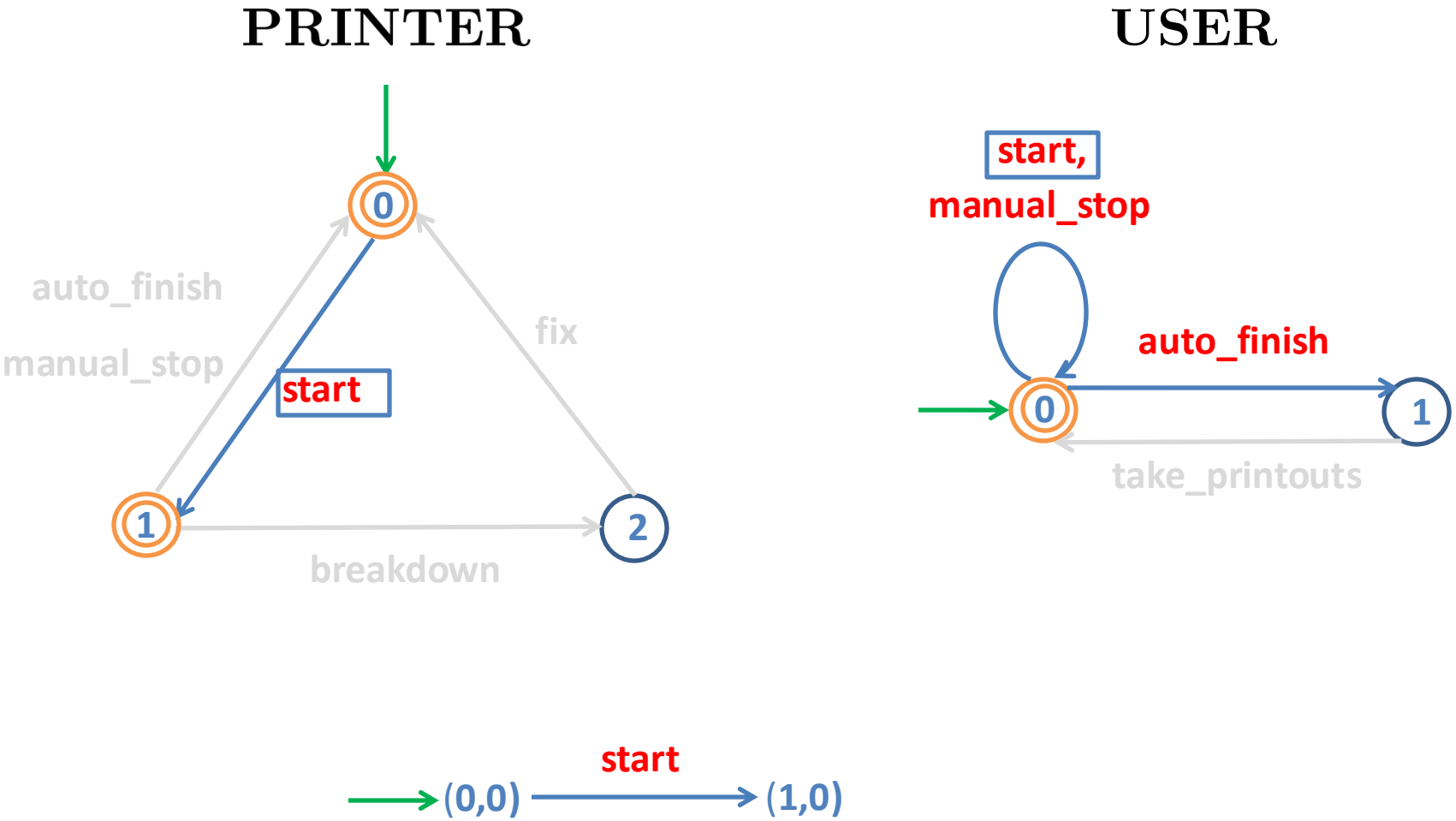
Printer and User



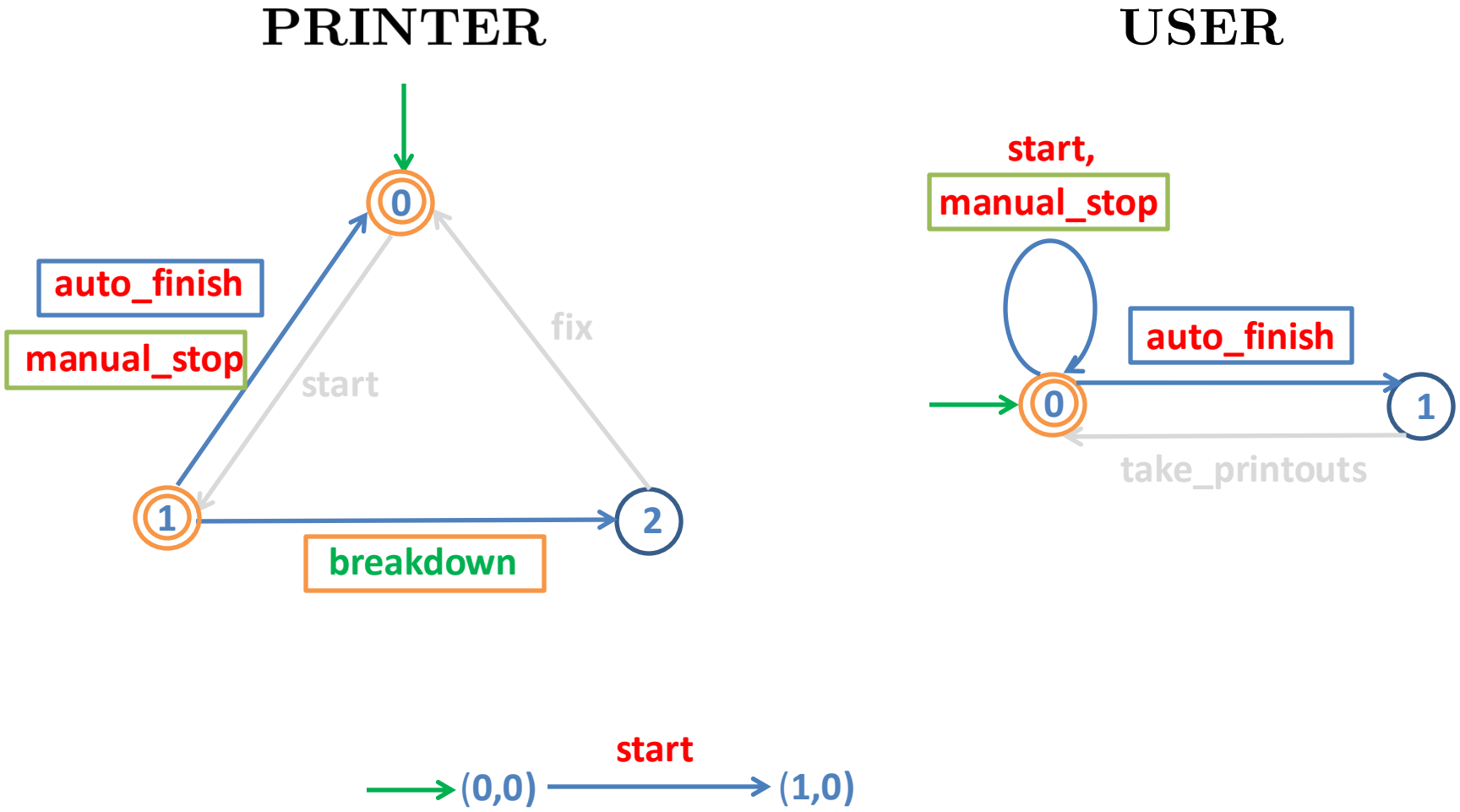
Printer and User



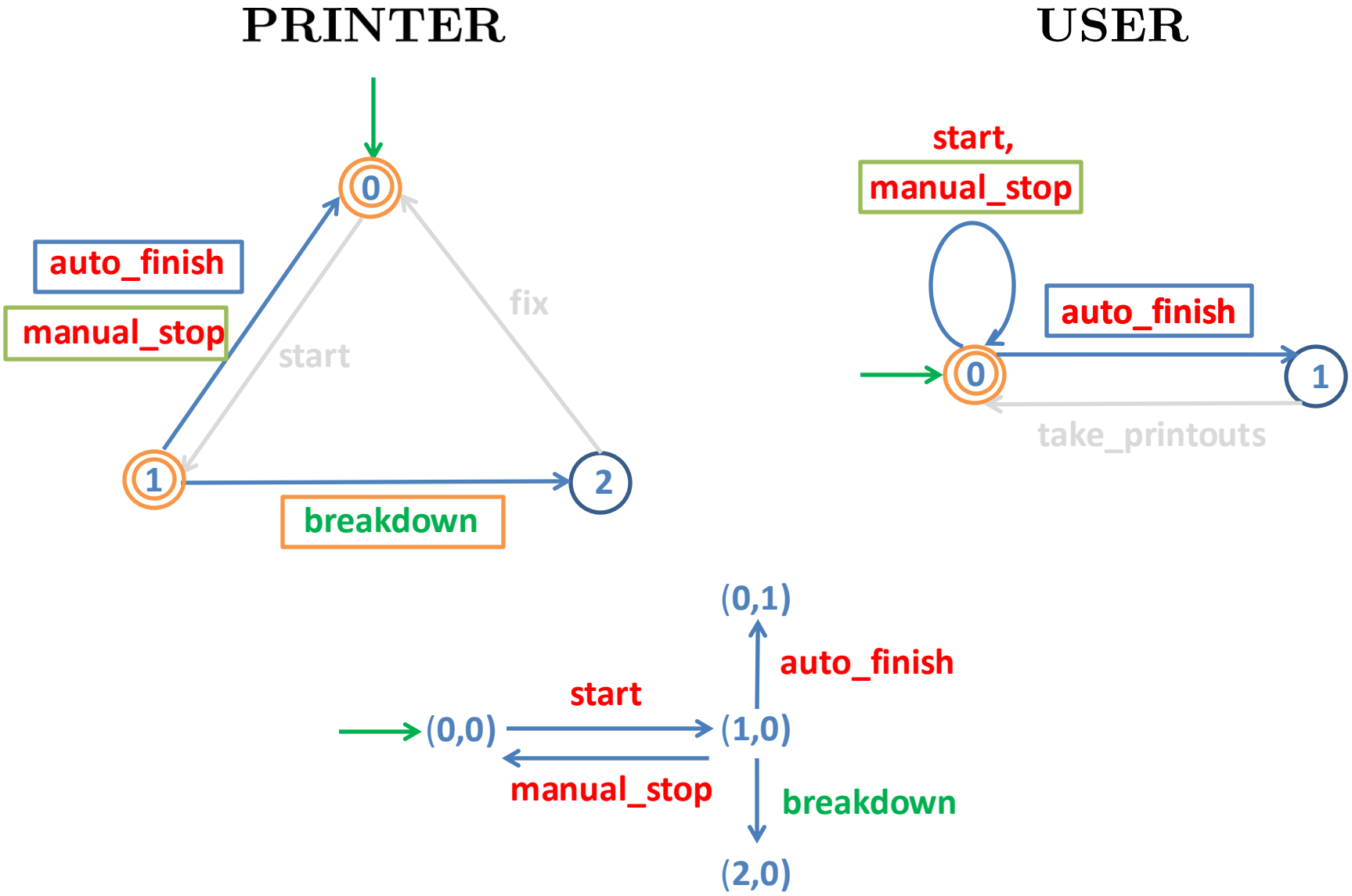
Printer and User



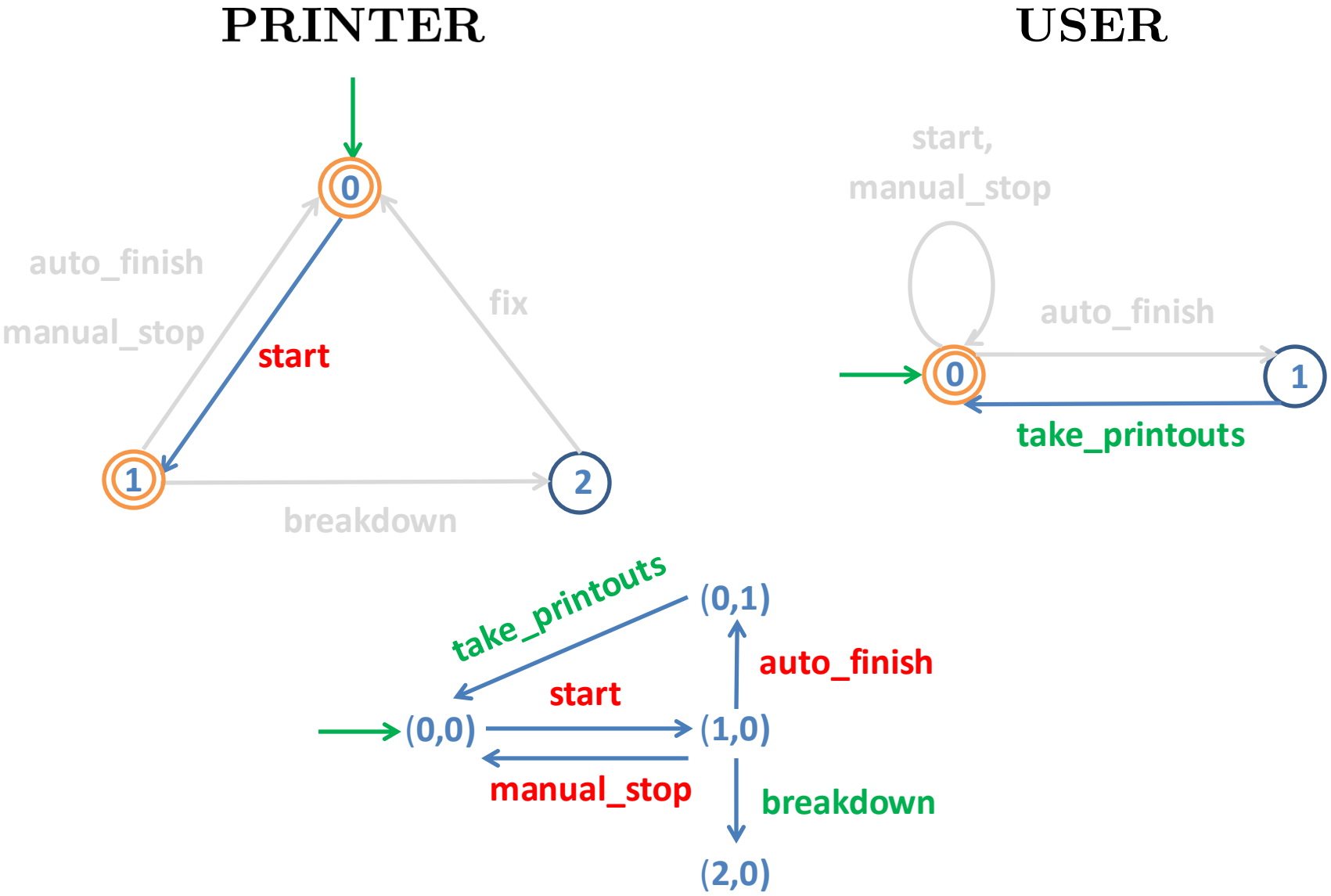
Printer and User



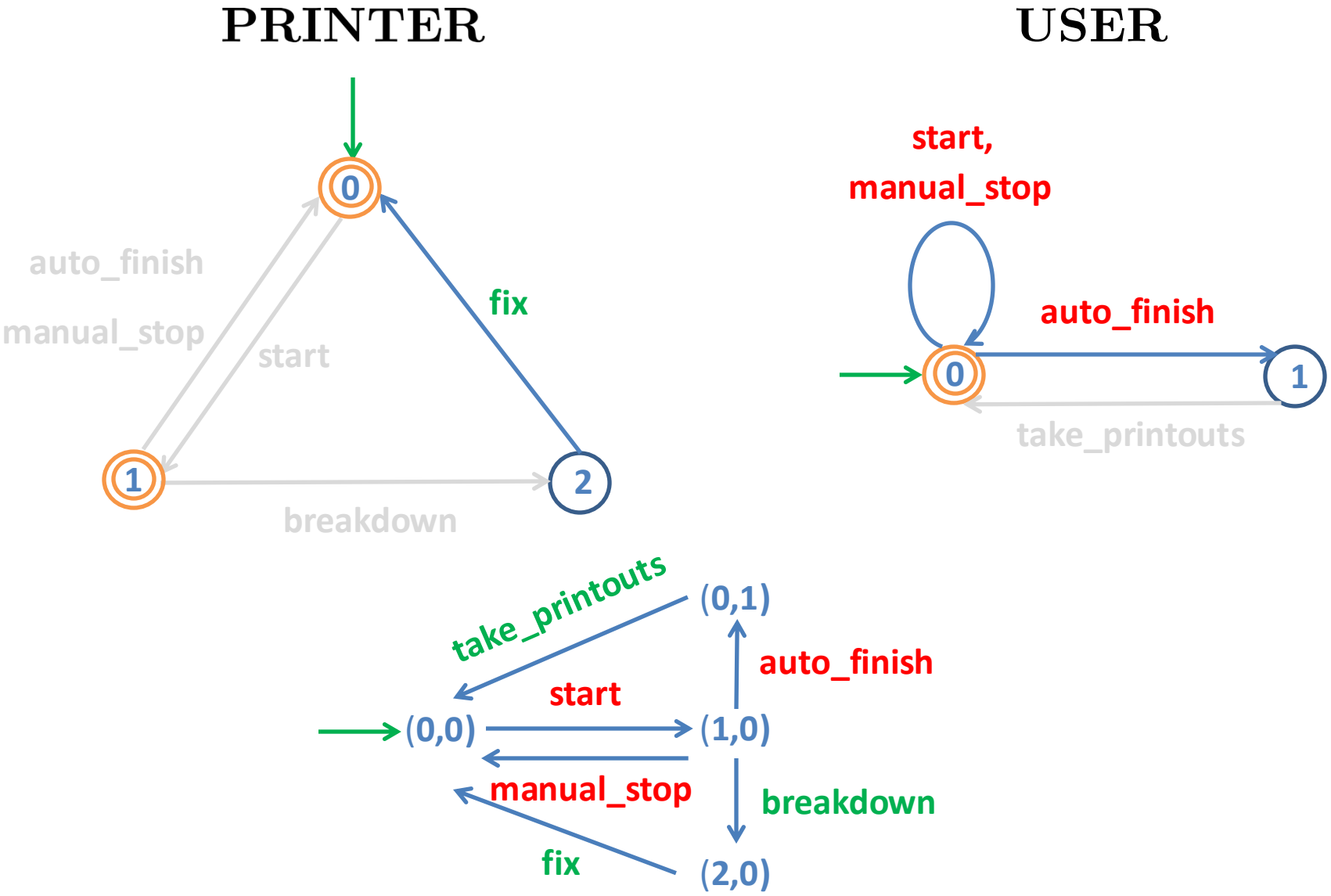
Printer and User



Printer and User



Printer and User



Synchronous product: definition

Let $\mathbf{G}_1 = (Q_1, \Sigma_1, \delta_1, q_{0,1}, Q_{m,1})$
and $\mathbf{G}_2 = (Q_2, \Sigma_2, \delta_2, q_{0,2}, Q_{m,2})$ be two automata

The synchronous product of $\mathbf{G}_1$ and $\mathbf{G}_2$, written $\mathbf{G}_1 \times \mathbf{G}_2$,
is a new automaton $\mathbf{G} = (Q, \Sigma, \delta, q_0, Q_m)$

where $Q = Q_1 \times Q_2 = \{(q_1, q_2) \mid q_1 \in Q_1 \ \& \ q_2 \in Q_2\}$,

$$\Sigma = \Sigma_1 \cup \Sigma_2,\ q_0 = (q_{0,1}, q_{0,2}),\ Q_m = Q_{m,1} \times Q_{m,2}$$

$$\delta : (Q_1 \times Q_2) \times \Sigma \to (Q_1 \times Q_2)$$

Synchronous product: definition

$\delta : (Q_1 \times Q_2) \times \Sigma \to (Q_1 \times Q_2)$

Case 1: $\sigma \in \Sigma_1 \setminus \Sigma_2$ and $\delta_1(q_1, \sigma)!$

$$\delta((q_1, q_2), \sigma) = (\delta_1(q_1, \sigma), q_2)$$

Case 2: $\sigma \in \Sigma_2 \setminus \Sigma_1$ and $\delta_2(q_2, \sigma)!$

$$\delta((q_1, q_2), \sigma) = (q_1, \delta_2(q_2, \sigma))$$

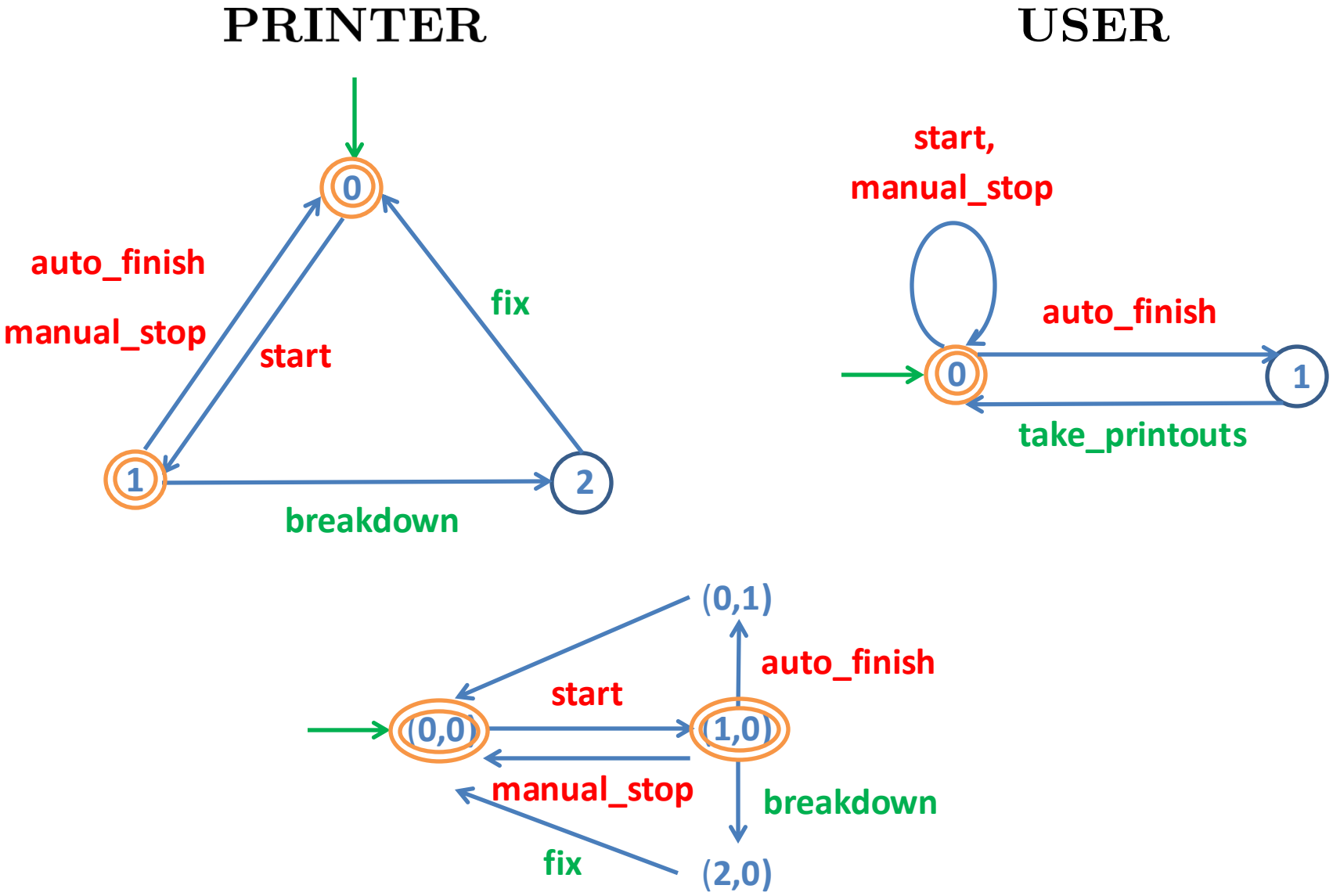
Case 3: $\sigma \in \Sigma_1 \cap \Sigma_2$, $\delta_1(q_1, \sigma)!$, and $\delta_2(q_2, \sigma)!$

$$\delta((q_1, q_2), \sigma) = (\delta_1(q_1, \sigma), \delta_2(q_2, \sigma))$$

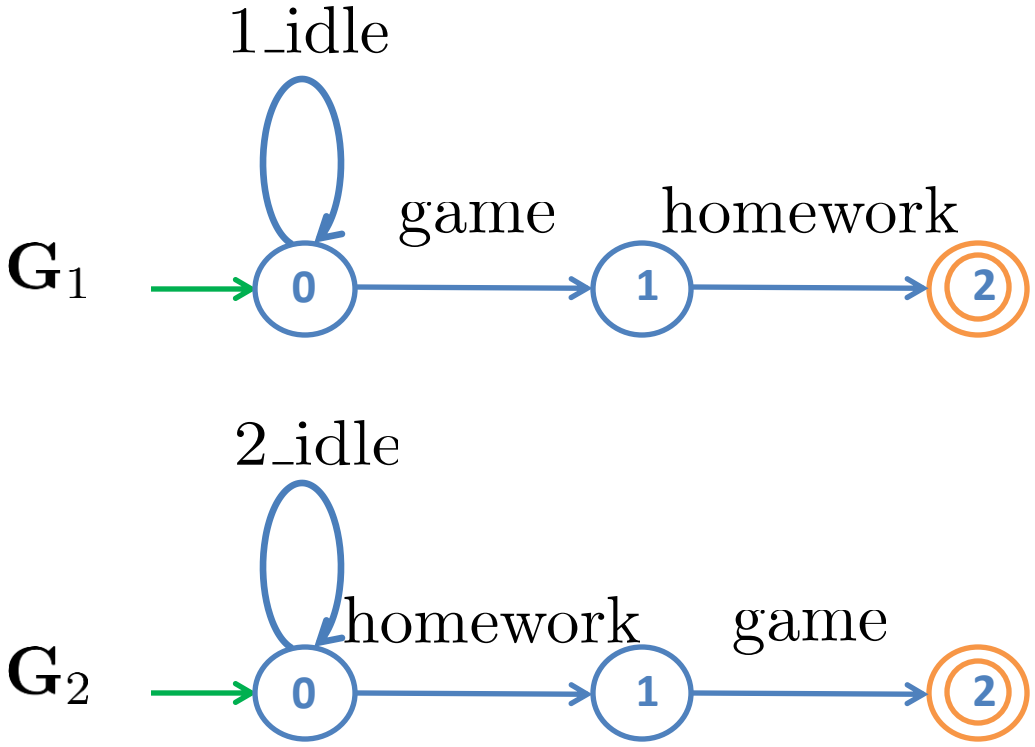
Case 4: otherwise

$$\neg\delta((q_1, q_2), \sigma)!$$

Printer and User



Event blocking



$\mathbf{G}_1 || \mathbf{G}_2$

Synchronously nonconflicting

Let $\mathbf{G}_1 = (Q_1, \Sigma_1, \delta_1, q_{0,1}, Q_{m,1})$
and $\mathbf{G}_2 = (Q_2, \Sigma_2, \delta_2, q_{0,2}, Q_{m,2})$ be two automata

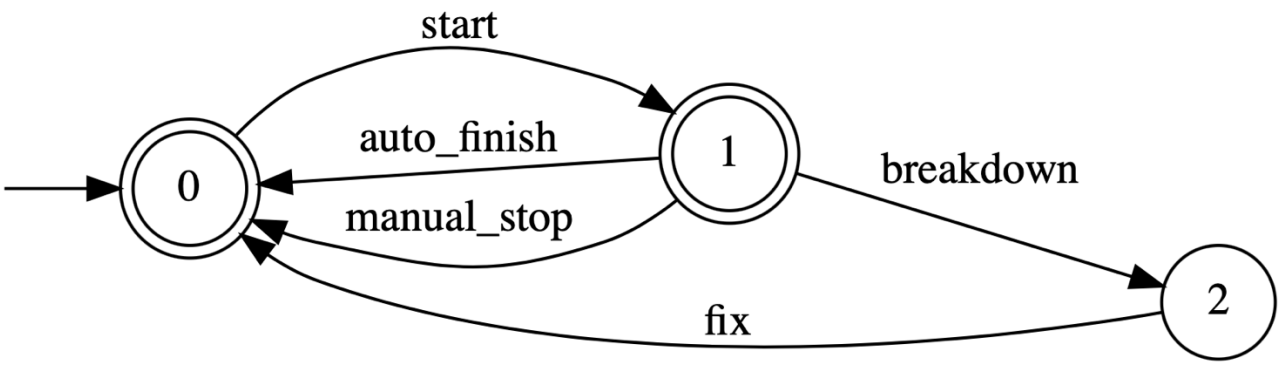
$\mathbf{G}_1$ and $\mathbf{G}_2$ are (synchronously) nonconflicting
if their synchronous product $\mathbf{G}_1||\mathbf{G}_2$ is nonblocking

PyTCT create

```
import pytct

pytct.init("mytct", overwrite=True)
```

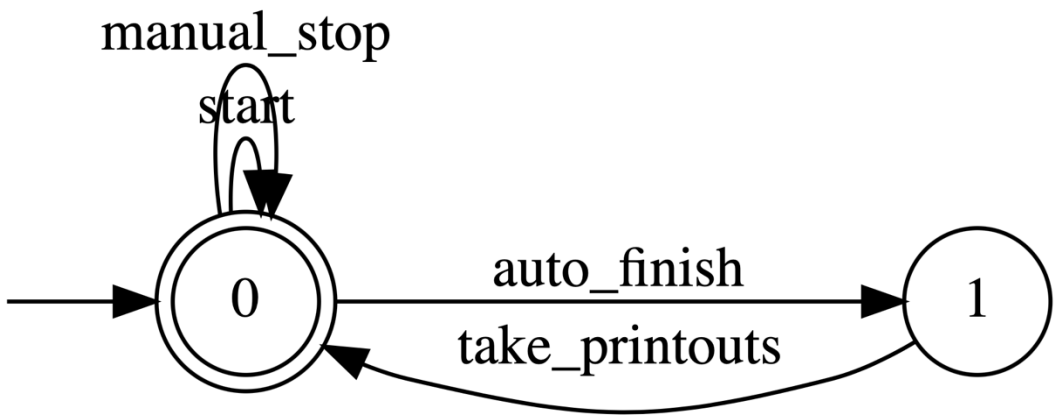
```
statenum=3 #number of states
#states are sequentially labeled 0,1,...,statenum
#initial state is labeled 0

trans=[(0,'start',1),
       (1,'auto_finish',0),
       (1,'manual_stop',0),
       (1,'breakdown',2),
       (2,'fix',0)] # set of transitions
#each triple is (exit state, event label, entering state)

marker = [0,1] #set of marker states

pytct.create('PRINTER', statenum, trans, marker)
#create automaton A
```

```
pytct.display_automaton('PRINTER')
```



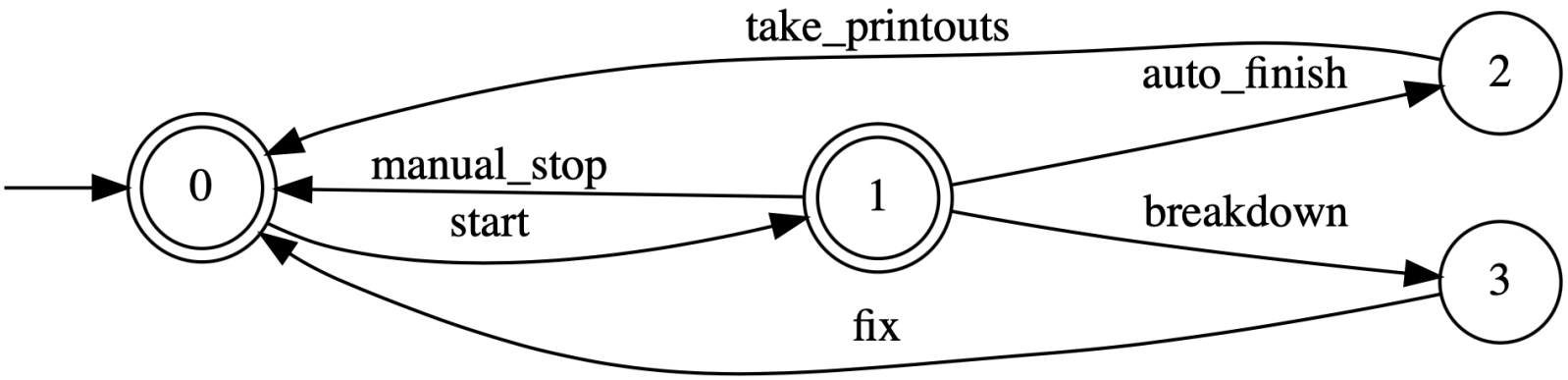
PyTCT create

```
statenum=2 #number of states
#states are sequentially labeled 0,1,...,statenum
#initial state is labeled 0

trans=[(0,'start',0),
       (0,'auto_finish',1),
       (0,'manual_stop',0),
       (1,'take_printouts',0)] # set of transitions
#each triple is (exit state, event label, entering state)

marker = [0] #set of marker states

pytct.create('HUMAN', statenum, trans, marker)
#create automaton A

pytct.display_automaton('HUMAN')
```



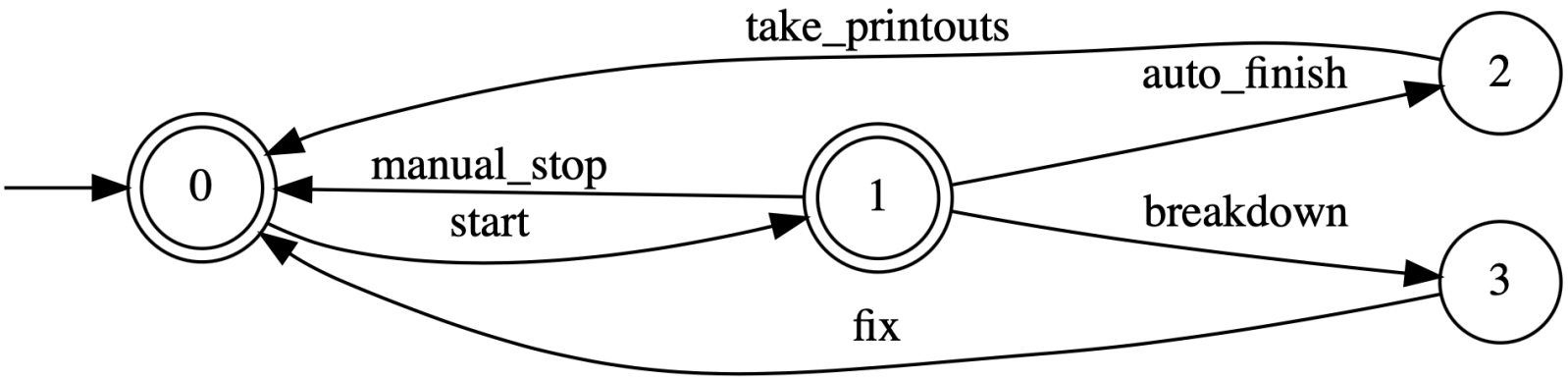
PyTCT sync

```
pytct.sync('PH_SYNC','PRINTER','HUMAN')

pytct.display_automaton('PH_SYNC')
```



PyTCT sync

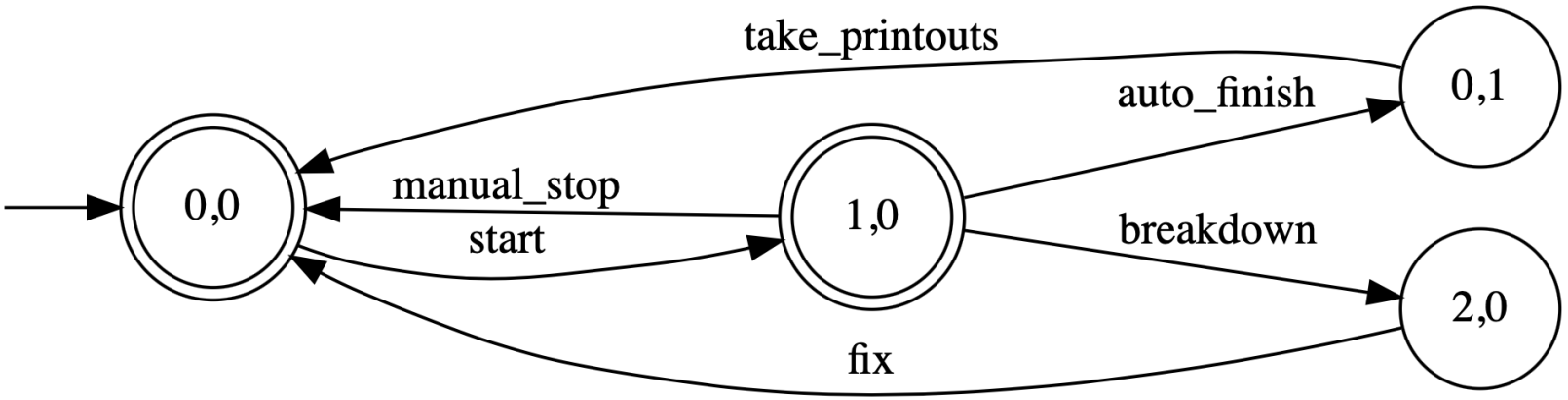
```
table = pytct.sync('PH','PRINTER','HUMAN',table=True)
print(table)
pytct.display_automaton('PH')
```




```
0: 0,0
1: 1,0
2: 0,1
3: 2,0
```

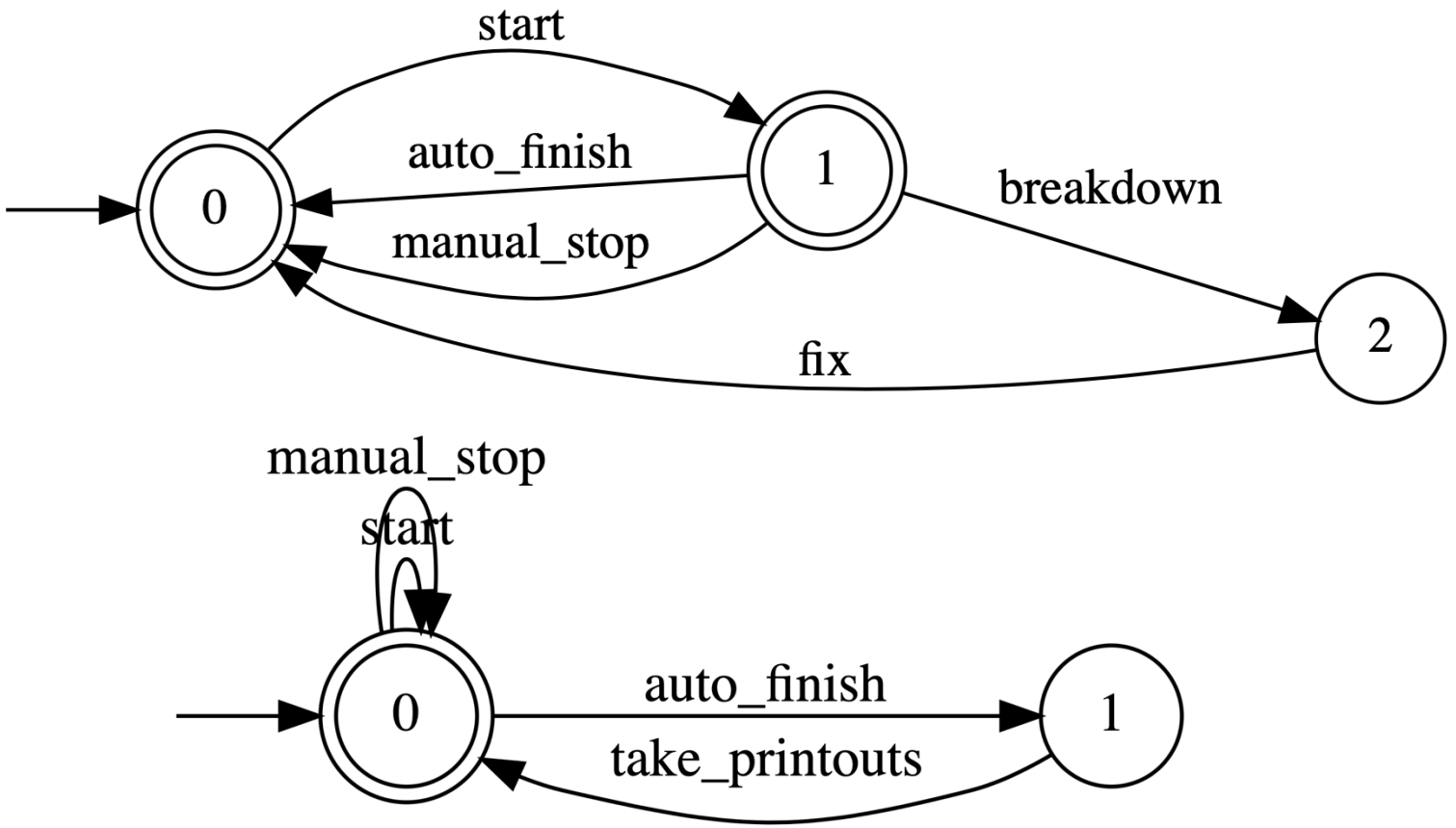
PyTCT sync

```
table = pytct.sync('PH','PRINTER','HUMAN',table=True, convert=True)
print(table)
pytct.display_automaton('PH')
```




```
0: 0,0
1: 1,0
2: 0,1
3: 2,0
```

PyTCT nonconflicting




```
pytct.sync('PH_SYNC','PRINTER','HUMAN')
```

```
pytct.is_nonblocking('PH_SYNC')
```

```
True
```

Recap

1. Create an automaton (create)
2. Properties of automaton (trim)
3. Synchronous product of automata (sync)
4. Feedback control loop
5. Controllability
6. Optimal supervisory control design

DES model

DES control

Feedback control loop

Supervisory control theory

Plant: target system to be controlled

modeled by automaton $\mathbf{P} = (X, \Sigma, \xi, x_0, X_m)$

control technology: parition Σ such that $\Sigma = \Sigma_c \dot{\cup} \Sigma_u$

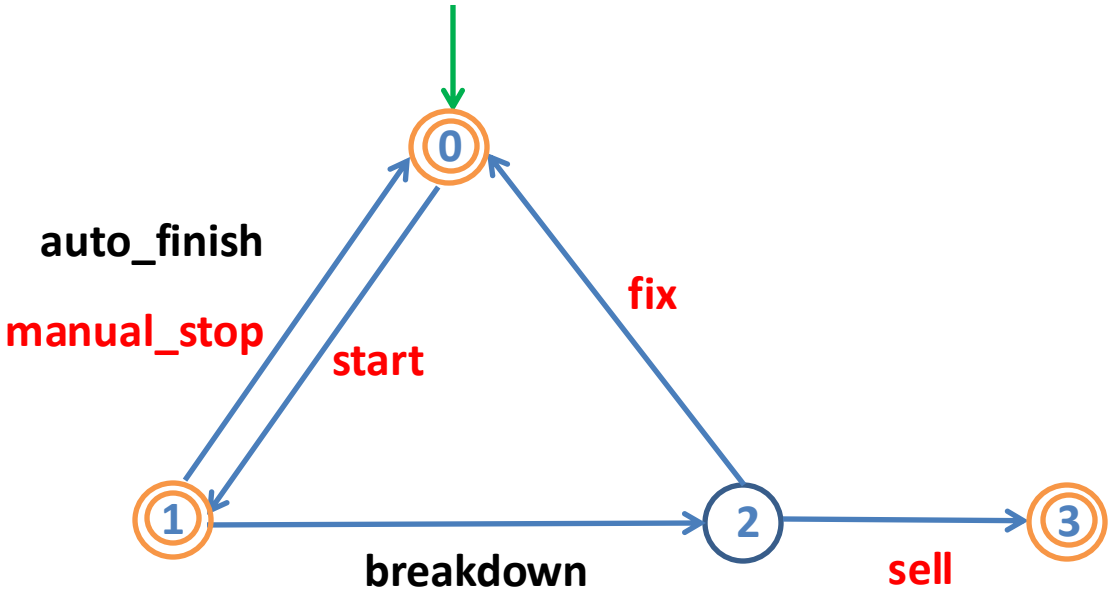
Σ_c: controllable event subset

Σ_u: uncontrollable event subset

Specification: control requirement

Supervisor: control rules that make plant satisfy specification

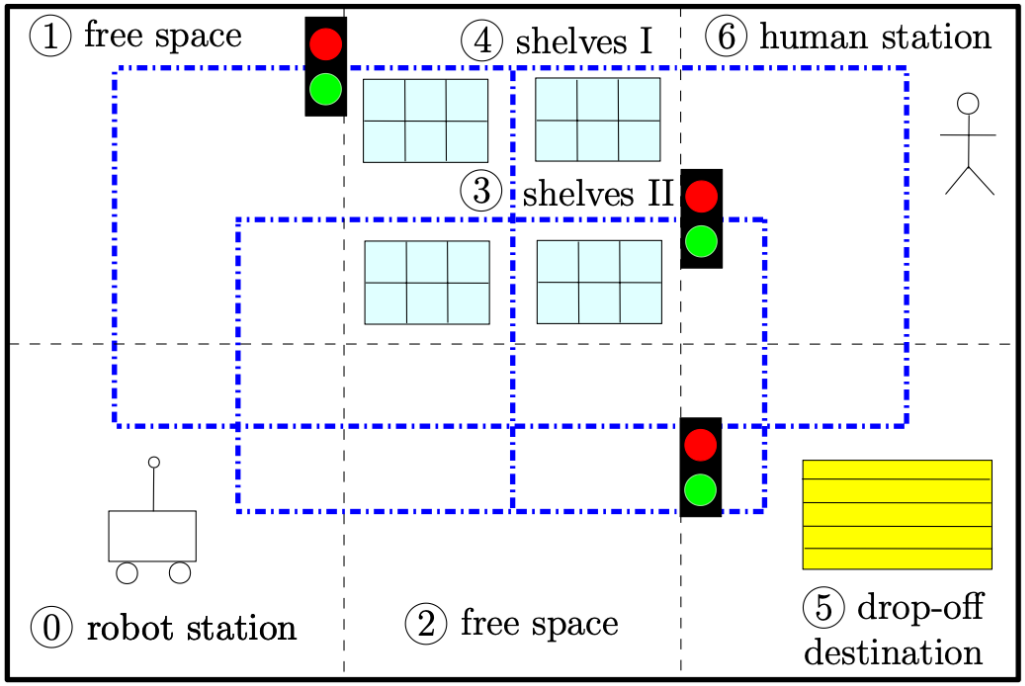
A printer with SOLD state

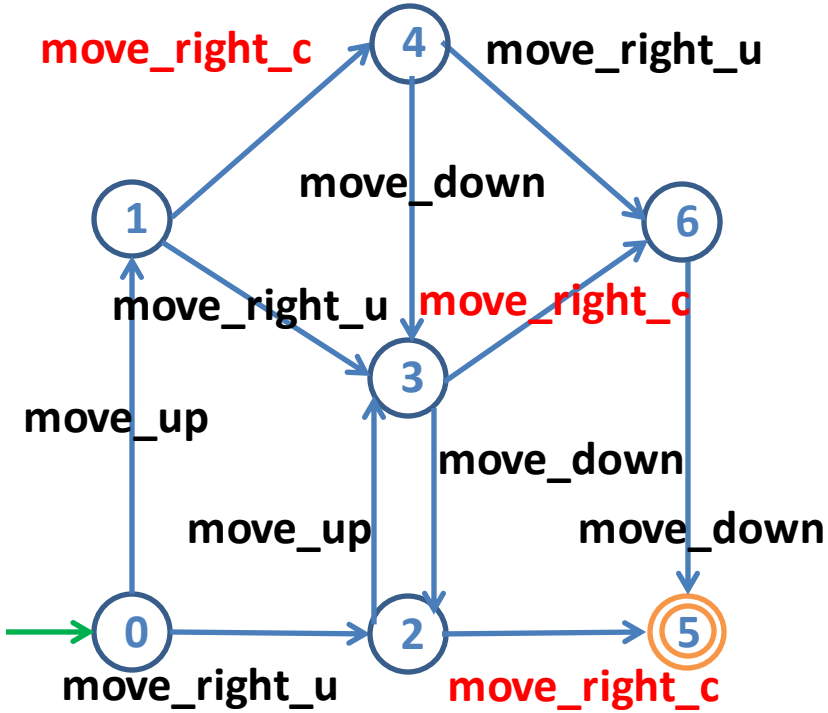


$$\Sigma_c = \{start, manual\_stop, fix, sell\}$$

$$\Sigma_u = \{auto\_finish, breakdown\}$$

A warehouse with traffic lights

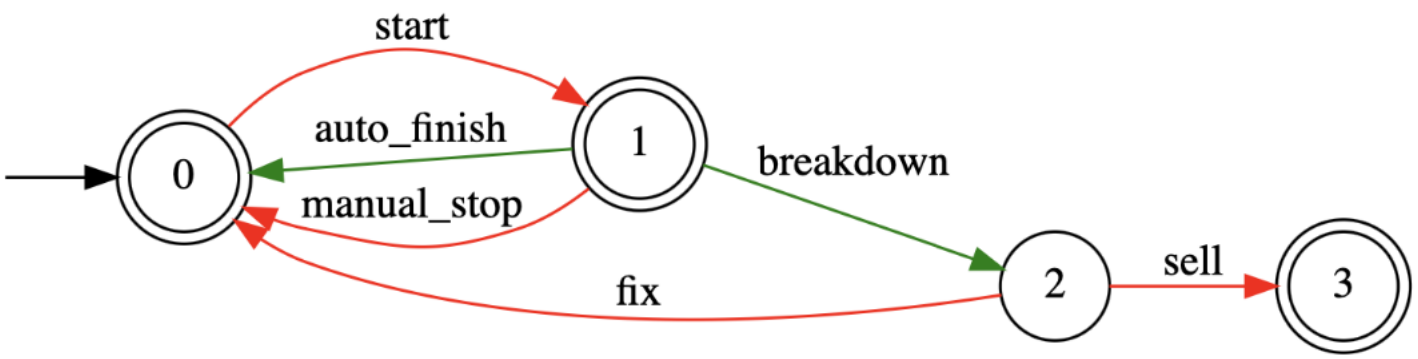




PyTCT create (controllable events)

```
statenum=4 #number of states
#states are sequentially labeled 0,1,...,statenum-1
#initial state is labeled 0

trans=[(0,'start',1,'c'),
       (1,'auto_finish',0,'u'),
       (1,'manual_stop',0,'c'),
       (1,'breakdown',2,'u'),
       (2,'fixe',0,'c')
       (2,'sell',3,'c')] #set of transitions
#each triple is (exit state, event label, entering state)
#each event is either 'c' (controllable) or 'u' (
    uncontrollable)

marker = [0,1,3] #set of marker states

pytct.create('PRINTER_SELL', statenum, trans, marker)
#create automaton PRINTER_SELL

pytct.display_automaton('PRINTER_SELL',color=True)
#plot PRINTER_SELL.DES with color coding
```

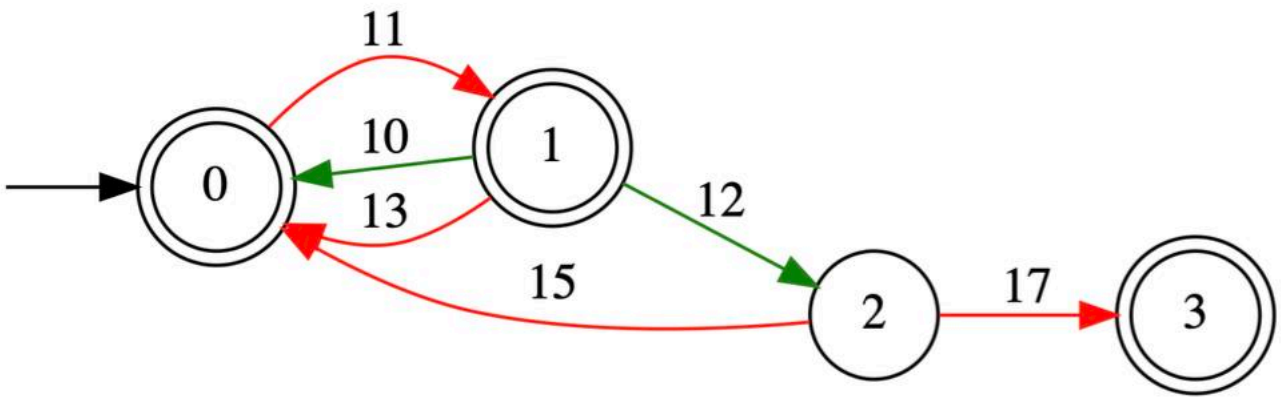
PyTCT display_automaton (color)



PyTCT create (controllable events)

```
statenum=4 #number of states
#states are sequentially labeled 0,1,...,statenum-1
#initial state is labeled 0

trans=[(0,11,1),
       (1,10,0),
       (1,13,0),
       (1,12,2),
       (2,15,0)] #set of transitions
#each triple is (exit state, event label, entering state)
#odd numbers for controllable events; even numbers for
    uncontrollable events

marker = [0,1,3] #set of marker states

pytct.create('PRINTER_SELL', statenum, trans, marker)
#create automaton PRINTER_SELL

pytct.display_automaton('PRINTER_SELL',color=True)
#plot PRINTER_SELL.DES with color coding
```

PyTCT display_automaton (color)



Specification: control requirement

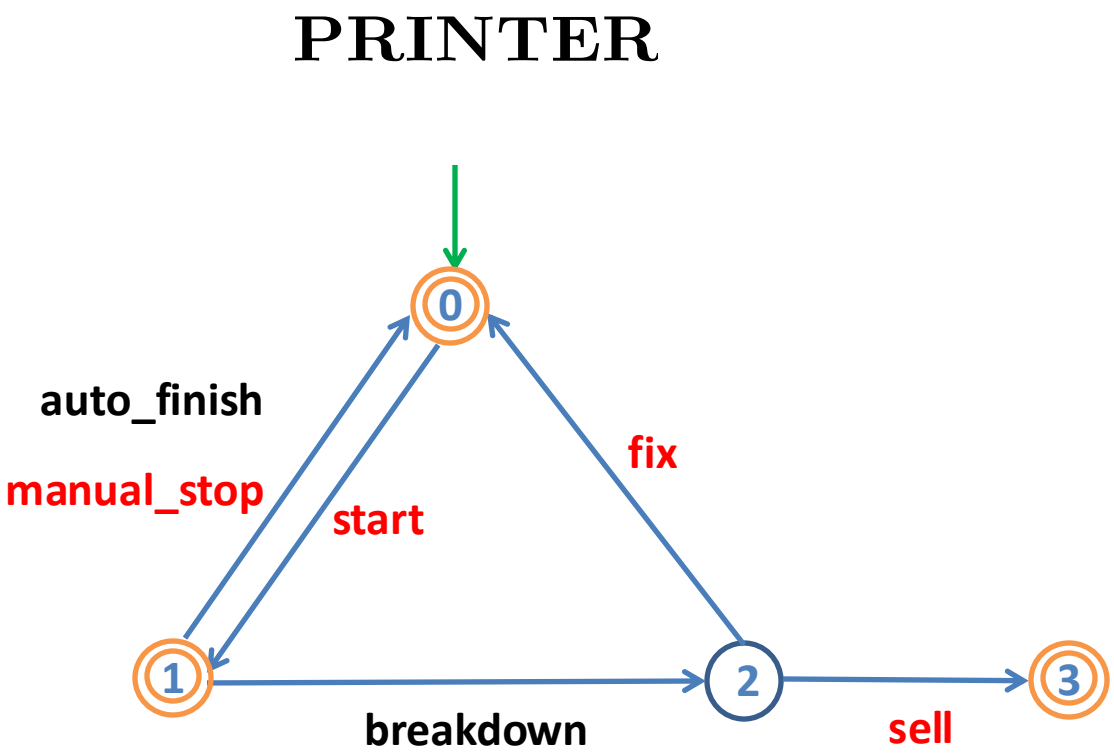
Plant $\mathbf{P} = (X, \Sigma, \xi, x_0, X_m)$

Specification $\mathbf{S}$: subautomaton of $\mathbf{P}$

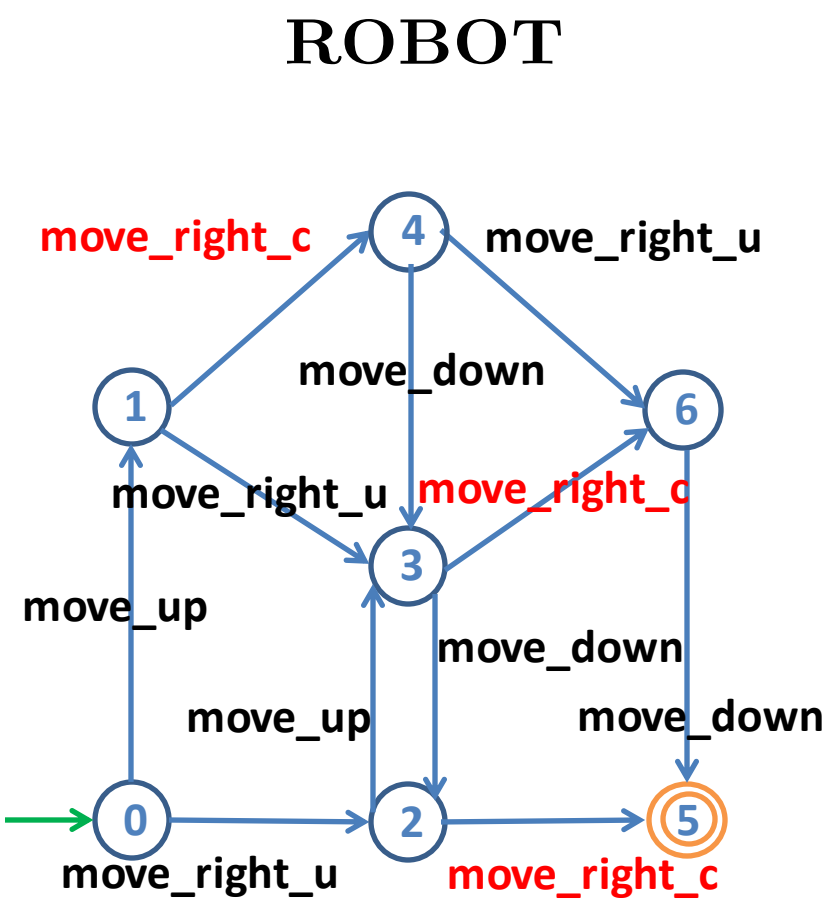
$$\mathbf{S} = (X_s, \Sigma_s, \xi_s, x_0, X_{s,m})$$

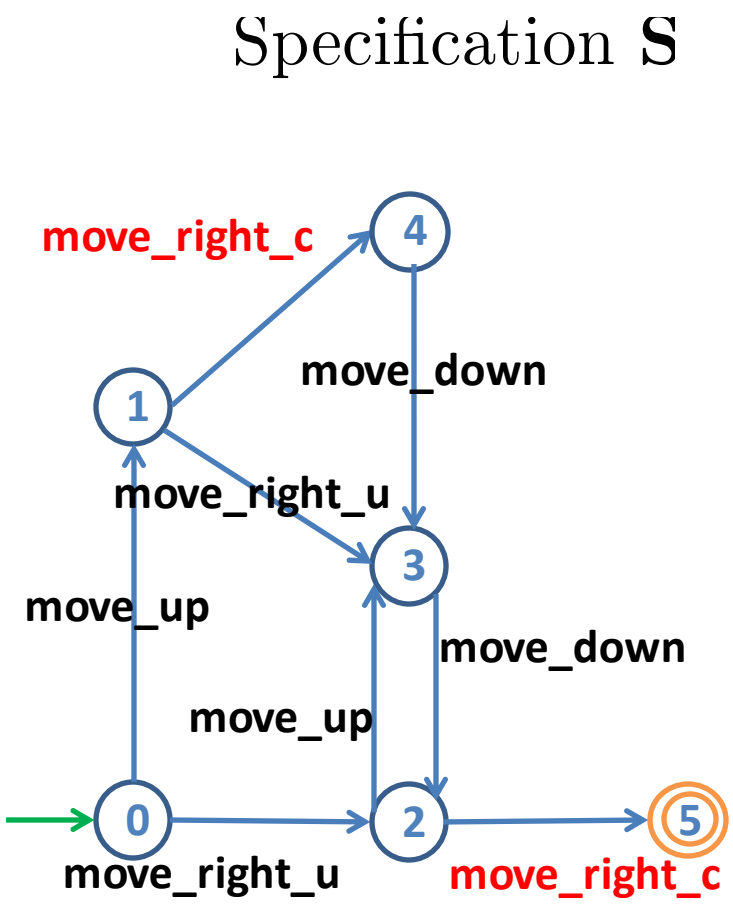
- $X_s \subseteq X$
- $\Sigma_s \subseteq \Sigma$
- $\xi_s \subseteq \xi$
- $X_{s,m} \subseteq X_m$

Specification: control requirement



Specification: control requirement





Supervisory controller

Plant $\mathbf{P} = (X, \Sigma, \xi, x_0, X_m)$

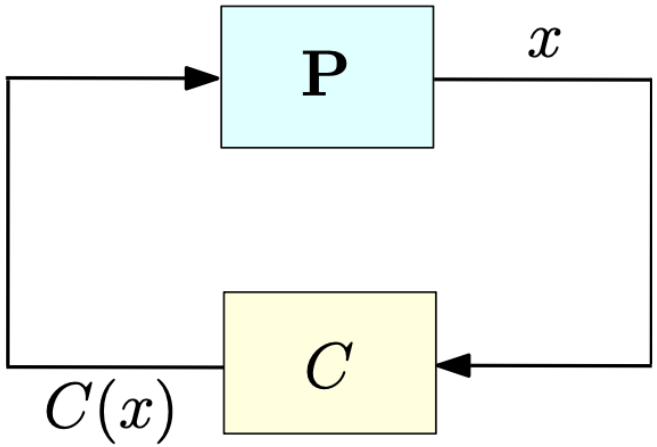
State-based supervisory controller $C : X \to 2^{\Sigma_c}$

(2^{Σ_c}: powerset of Σ_c)

Feedback control loop

Plant $\mathbf{P} = (X, \Sigma, \xi, x_0, X_m)$

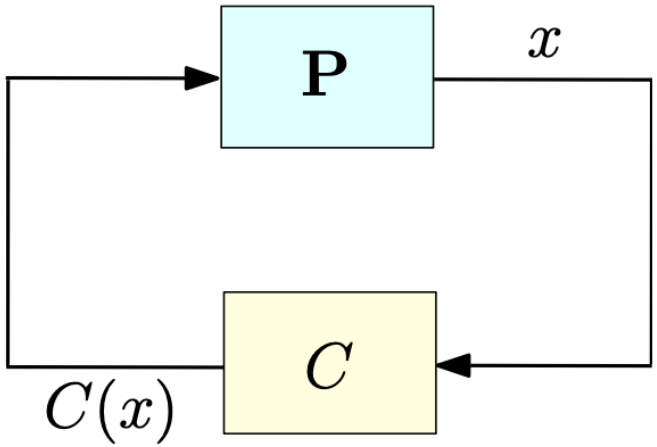
State-based supervisory controller $C : X \to 2^{\Sigma_c}$



Closed-loop system

Plant $\mathbf{P} = (X, \Sigma, \xi, x_0, X_m)$

Closed-loop system $C/\mathbf{P}$: subautomaton of $\mathbf{P}$



Closed-loop system

Plant $\mathbf{P} = (X, \Sigma, \xi, x_0, X_m)$

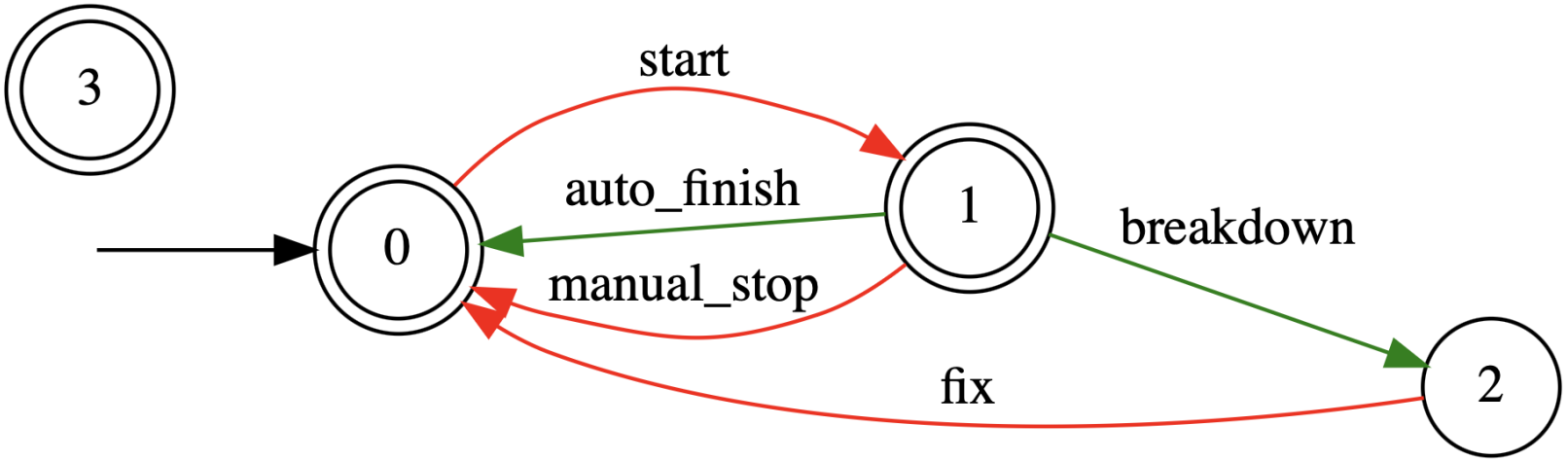
Closed-loop system $C/\mathbf{P}$: subautomaton of $\mathbf{P}$

Specification $\mathbf{S} = (X_s, \Sigma_s, \xi_s, x_0, X_{s,m})$ is
also a subautomaton of $\mathbf{P}$

Key question: *does there exist* C *s.t.* $C/\mathbf{P} = \mathbf{S}$?

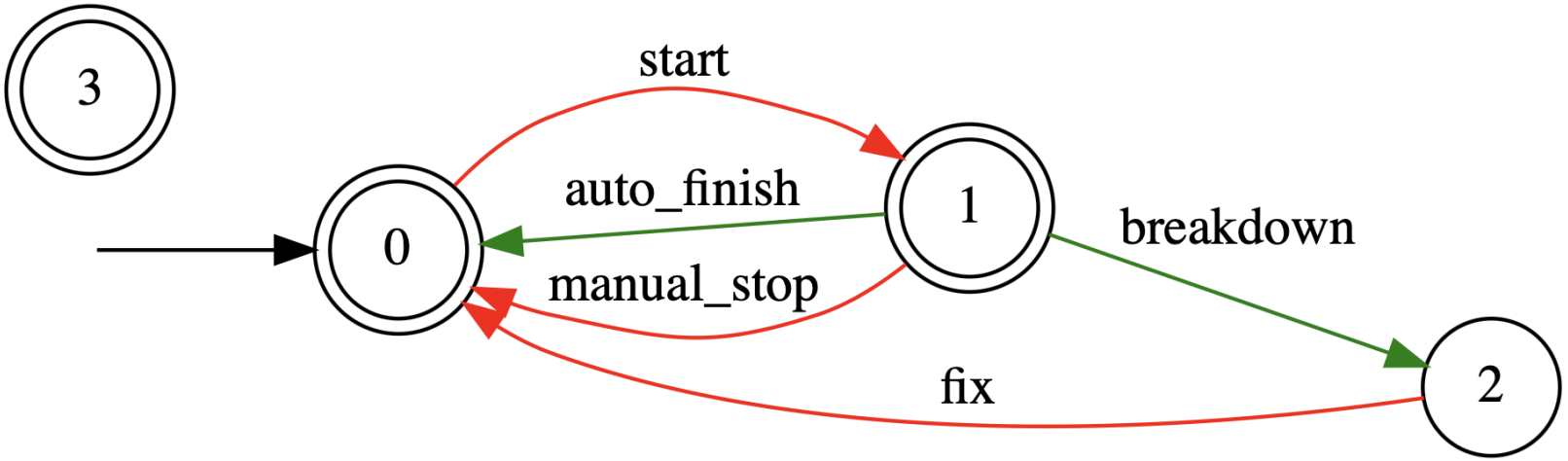
PyTCT subautomaton

```
pytct.subautomaton('S1','PRINTER_SELL',[3],[])
#create subautomaton S1 by removing from PRINTER_SELL
# [state list] and [transition list]

pytct.display_automaton('S1',color=True)
#plot S1.DES with color coding
```



PyTCT subautomaton

```
pytct.subautomaton('S2','PRINTER_SELL',[],[(2,'sell',3)])
#create subautomaton S2 by removing from PRINTER_SELL
# [state list] and [transition list]

pytct.display_automaton('S2',color=True)
#plot S2.DES with color coding
```



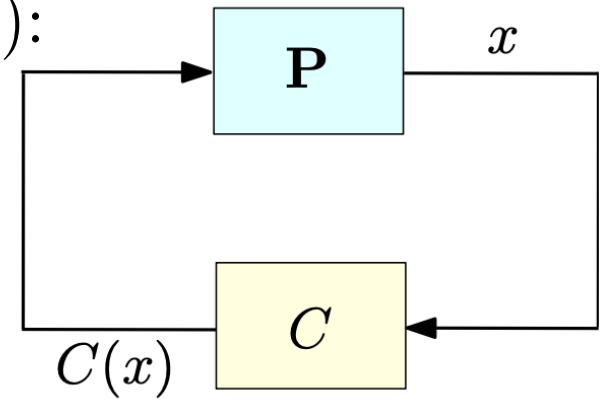
Controllability

Closed-loop system

Plant $\mathbf{P} = (X, \Sigma, \xi, x_0, X_m)$, $\Sigma = \Sigma_c \cup \Sigma_u$

Specification $\mathbf{S} = (X_s, \Sigma_s, \xi_s, x_0, X_{s,m})$:
subautomaton of $\mathbf{P}$

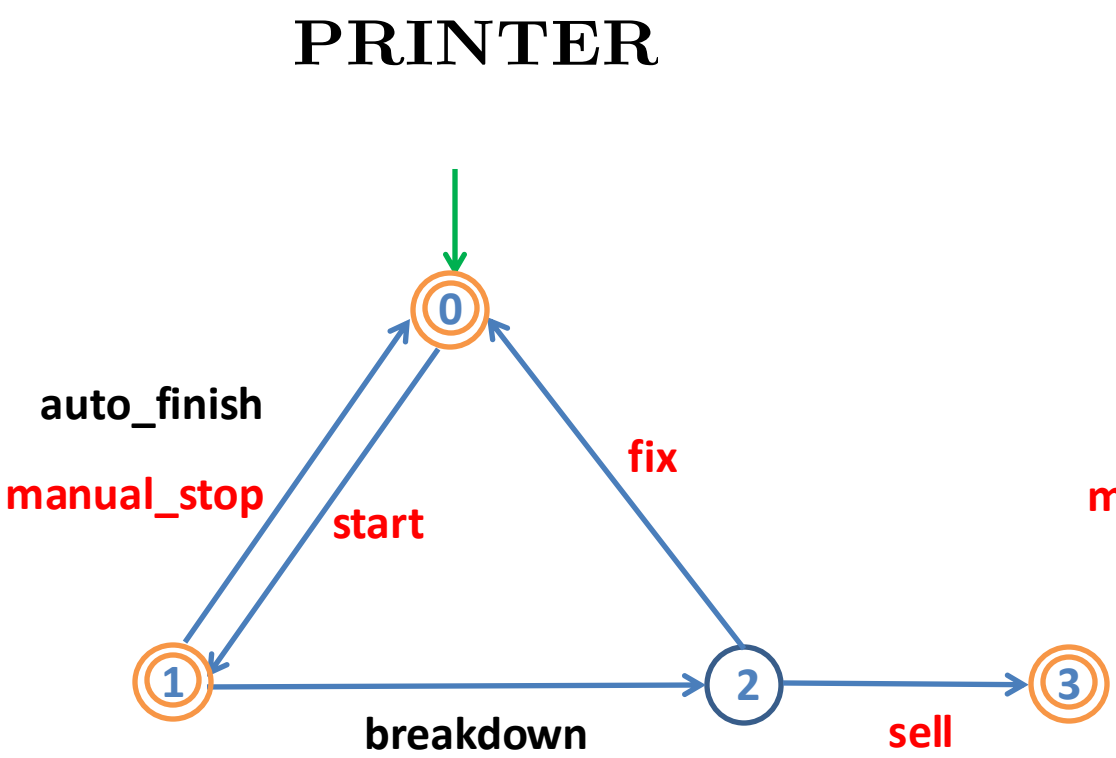
Supervisory controller $C : X \to 2^{\Sigma_c}$

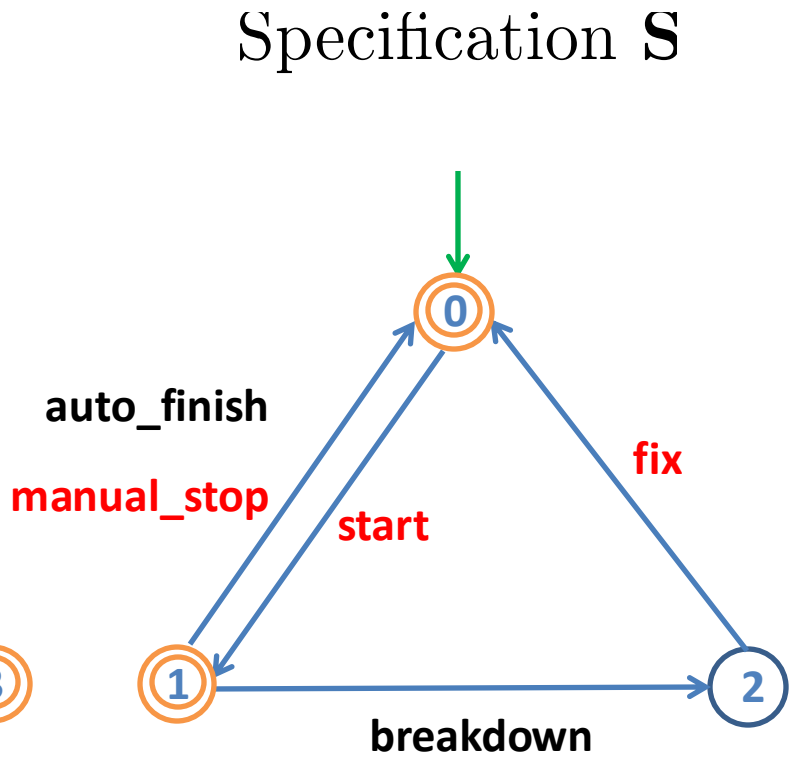


Closed-loop system $C/\mathbf{P}$: subautomaton of $\mathbf{P}$

Key question: *does there exist* C *s.t.* $C/\mathbf{P} = \mathbf{S}$?

A printer

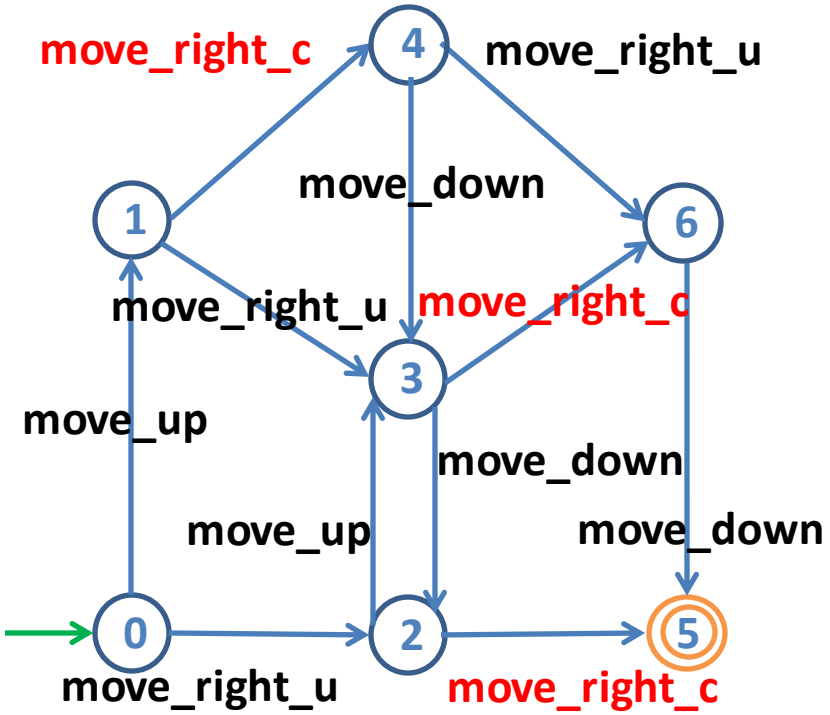




$$C(2) = \{sell\}$$

A warehouse robot

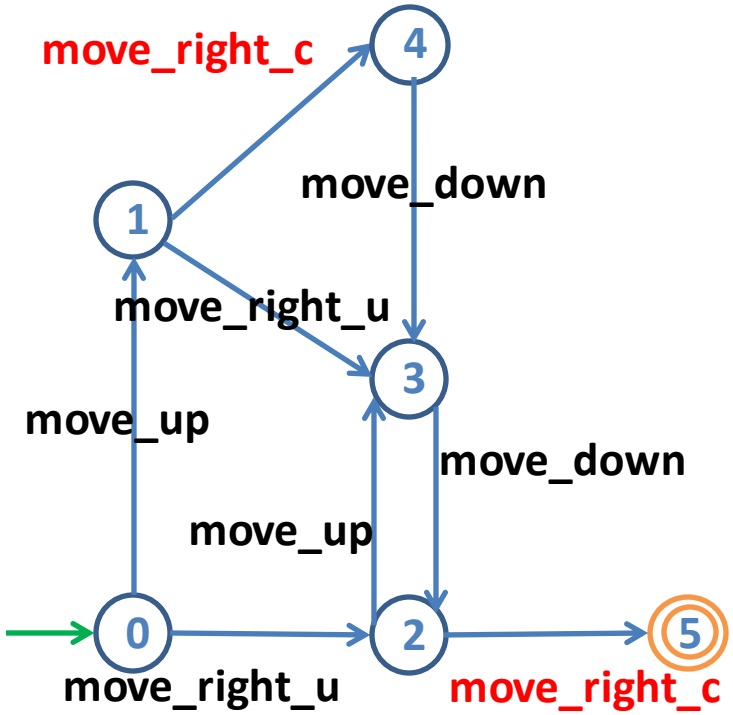
ROBOT



$C(3) = \{move\_right\_c\}$

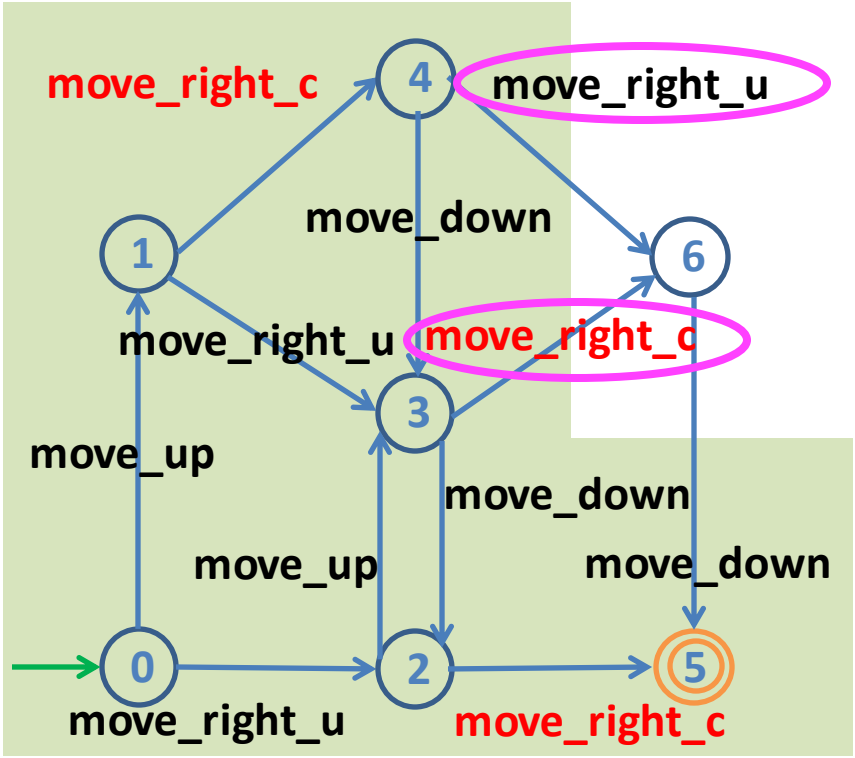
$C(4) = \{move\_right\_u\}$ (?)

Specification **S**



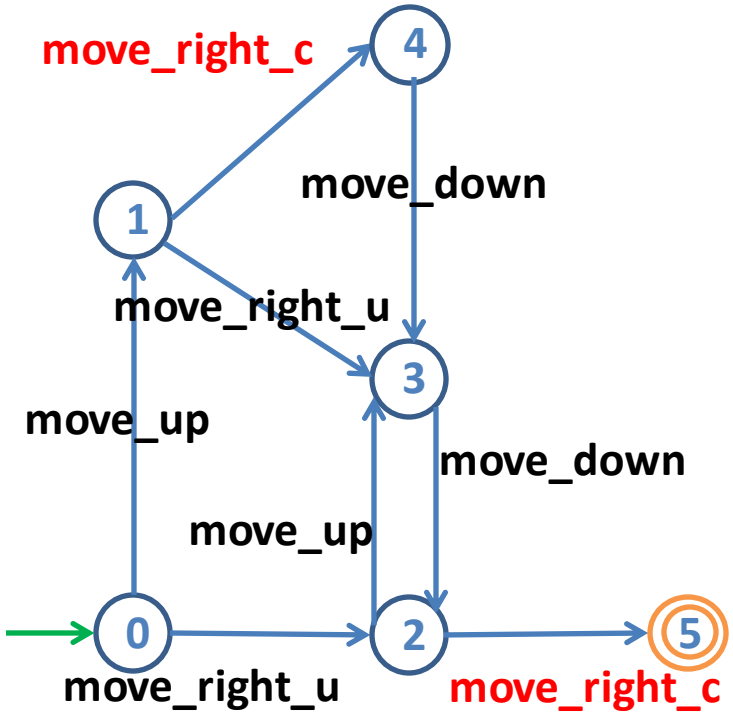
A warehouse robot

ROBOT



X_s

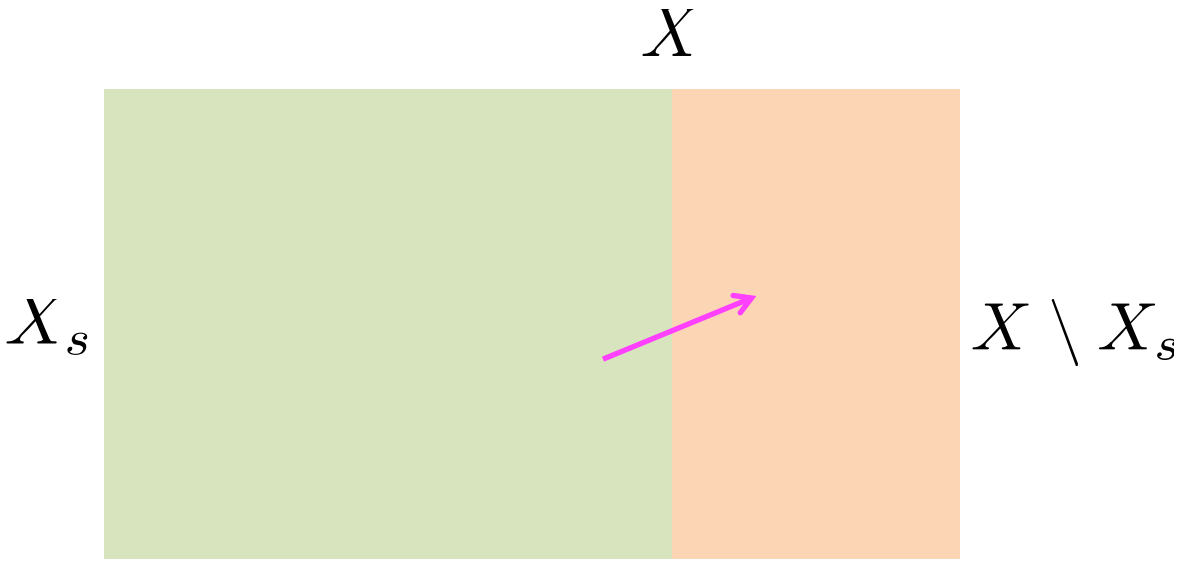
Specification **S**



Controllability

Plant $\mathbf{P} = (X, \Sigma, \xi, x_0, X_m)$, $\Sigma = \Sigma_c \cup \Sigma_u$

Specification $\mathbf{S} = (X_s, \Sigma_s, \xi_s, x_0, X_{s,m})$: subautomaton of $\mathbf{P}$

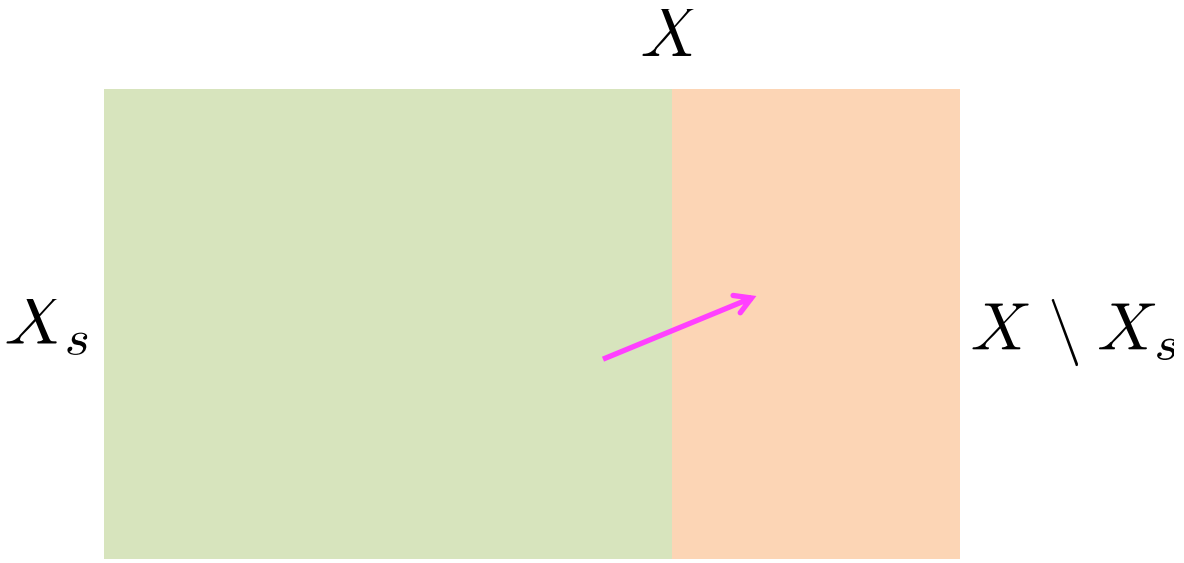


Key point: *is there an uncontrollable event existing X_s?*

Controllability

Plant $\mathbf{P} = (X, \Sigma, \xi, x_0, X_m)$, $\Sigma = \Sigma_c \cup \Sigma_u$

Specification $\mathbf{S} = (X_s, \Sigma_s, \xi_s, x_0, X_{s,m})$: subautomaton of $\mathbf{P}$

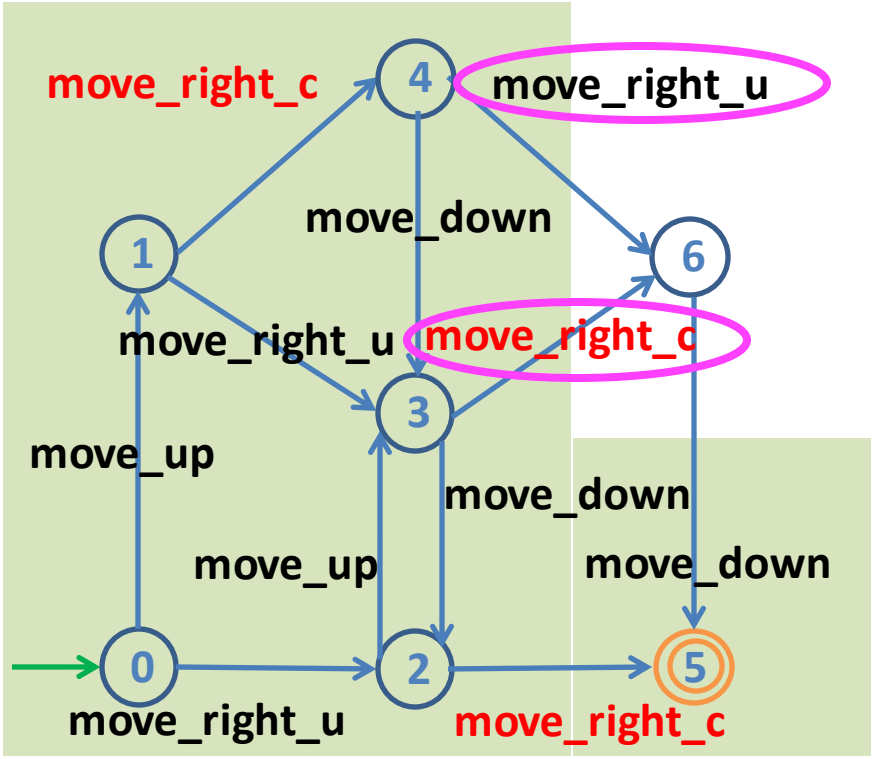


$\mathbf{S}$ is controllable (wrt. $\mathbf{P}$):

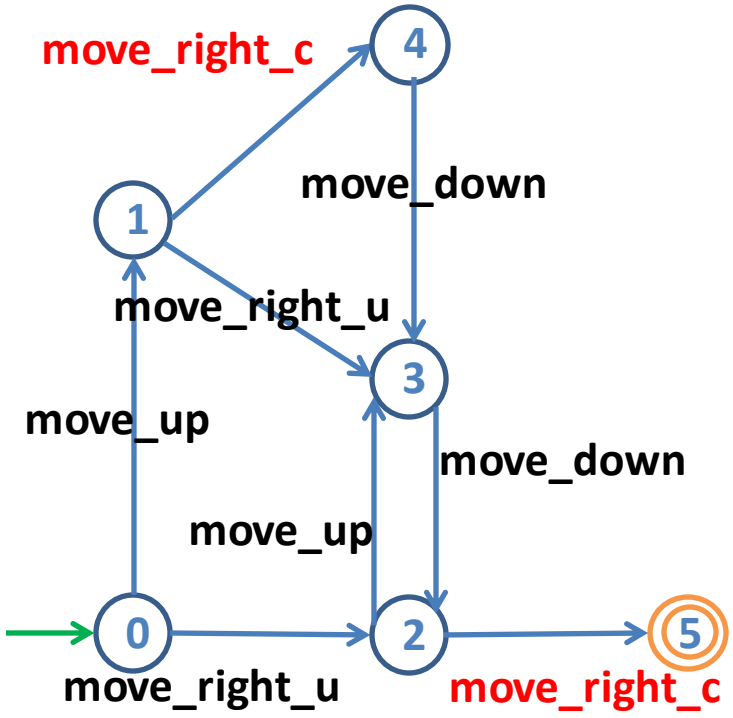
$(\forall x \in X_s, \forall \sigma_u \in \Sigma_u)\ \xi(x, \sigma_u)! \Rightarrow \xi_s(x, \sigma_u)!$

A warehouse robot

ROBOT

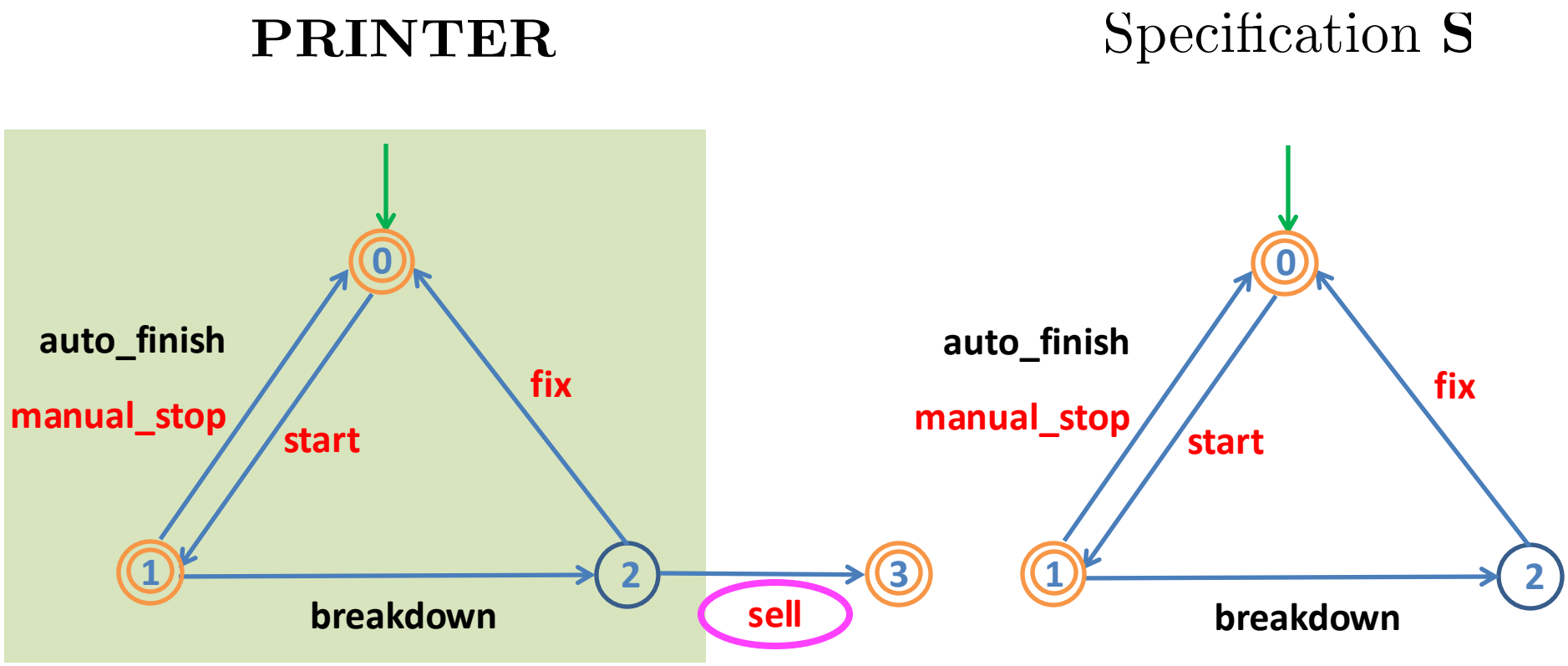


Specification **S**



S is not controllable

A printer

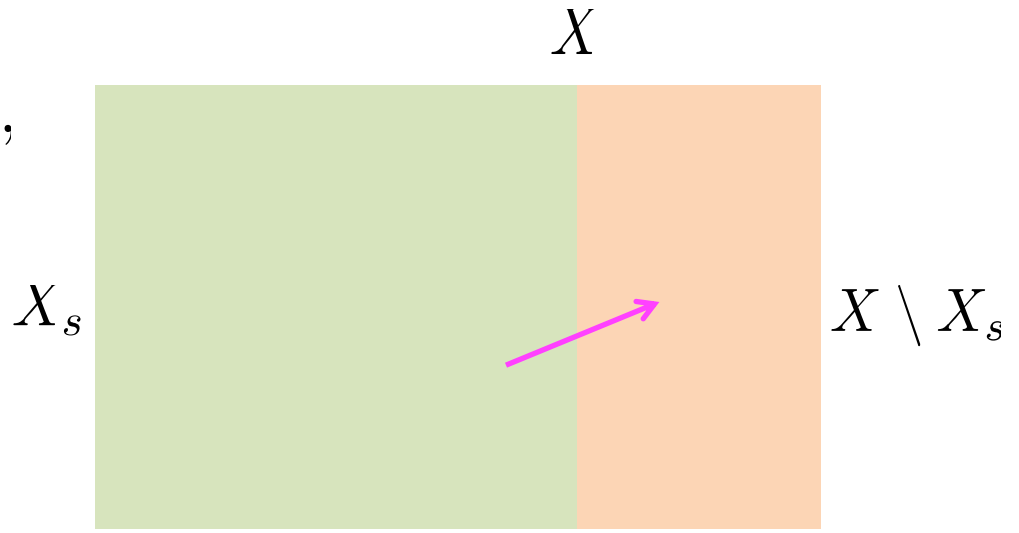


S is controllable

Controllability

Plant $\mathbf{P} = (X, \Sigma, \xi, x_0, X_m)$,
$\Sigma = \Sigma_c \cup \Sigma_u$

Specification
$\mathbf{S} = (X_s, \Sigma_s, \xi_s, x_0, X_{s,m})$

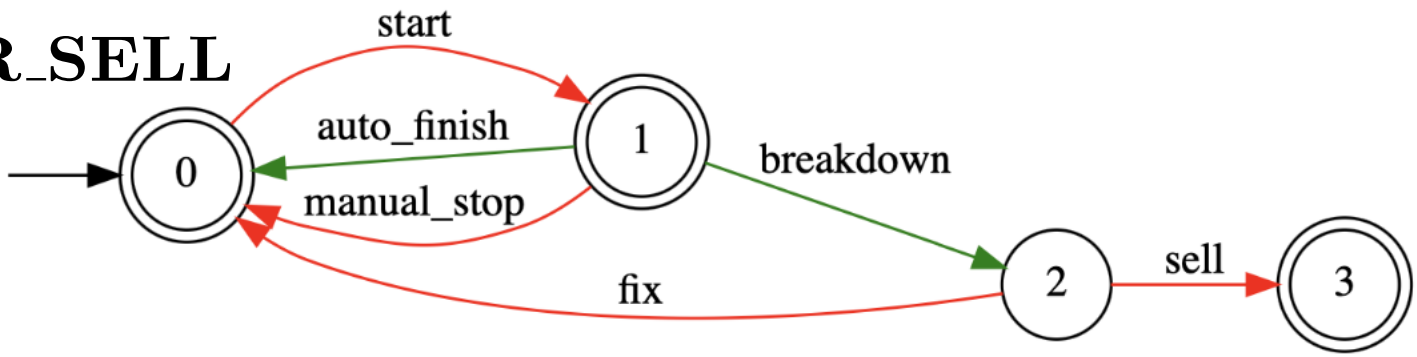


$\mathbf{S}$ is controllable (wrt. $\mathbf{P}$) if and only if
there exists a controller $C : X \to 2^{\Sigma_c}$ s.t. $C/\mathbf{P} = \mathbf{S}$

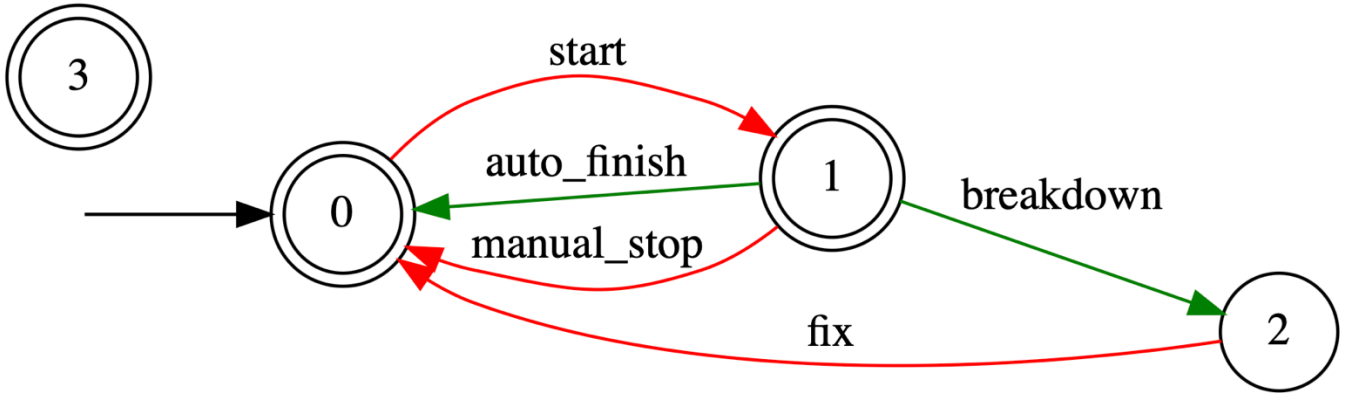
$$C(x) = \begin{cases} \{\sigma \in \Sigma_c \mid \xi(x,\sigma)! \ \& \ \neg\xi_s(x,\sigma)!\} & \text{if } x \in X_s \\ \emptyset & \text{if } x \in X \setminus X_s \end{cases}$$

PyTCT is_controllable

PRINTER_SELL



S

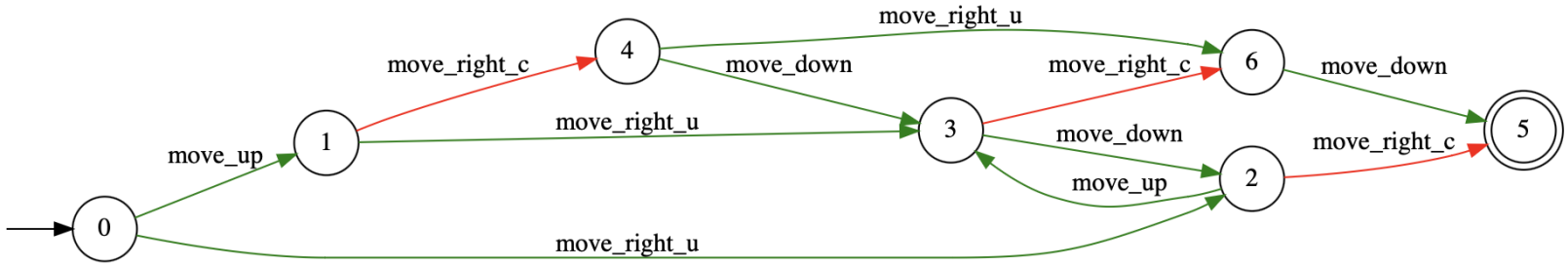



```
pytct.is_controllable('PRINTER_SELL','S')
#check if S is controllable wrt. PRINTER_SELL
```

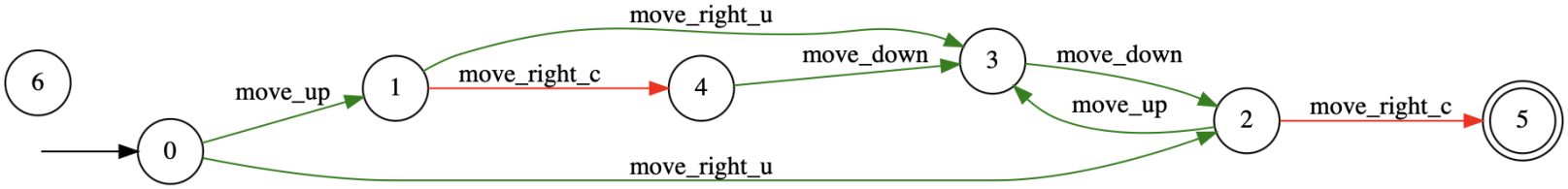
```
True
```

PyTCT is_controllable

ROBOT



S

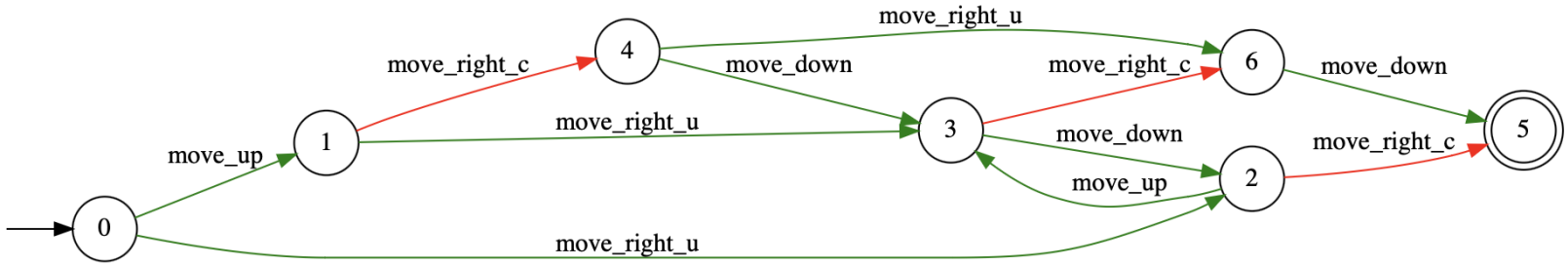



```
pytct.is_controllable('ROBOT','S')
#check if S is controllable wrt. ROBOT
```

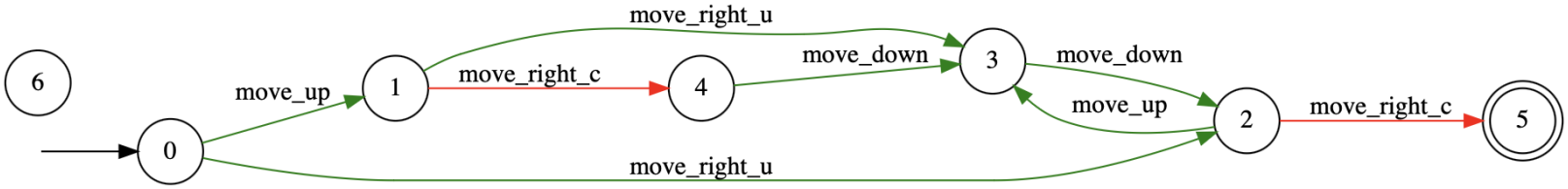
```
False
```

PyTCT uncontrollable_states

ROBOT



S

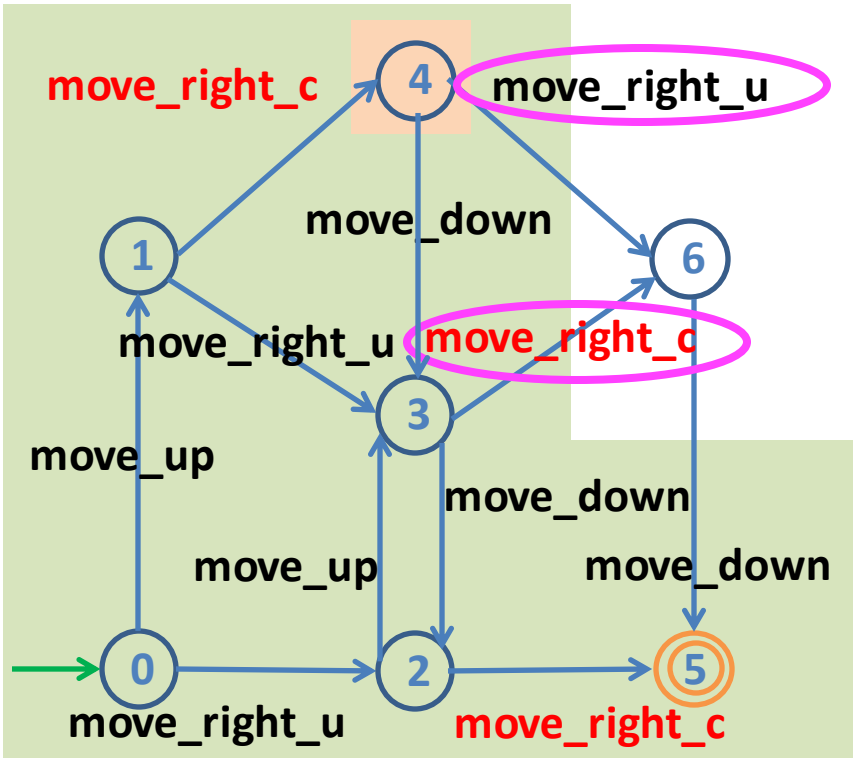



```
pytct.uncontrollable_states('ROBOT','S')
#compute the set of uncontrollable states
```

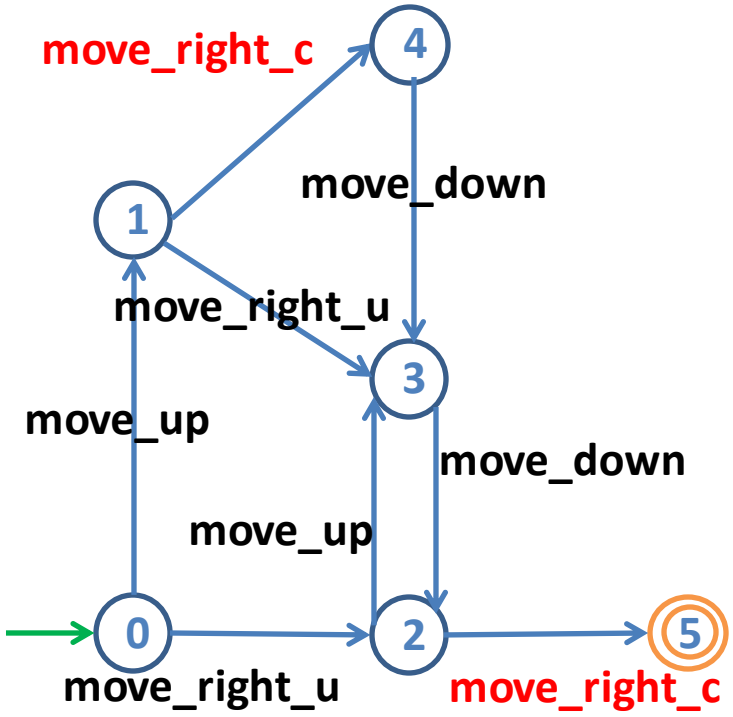
Optimal Supervisory Control

A warehouse robot

ROBOT



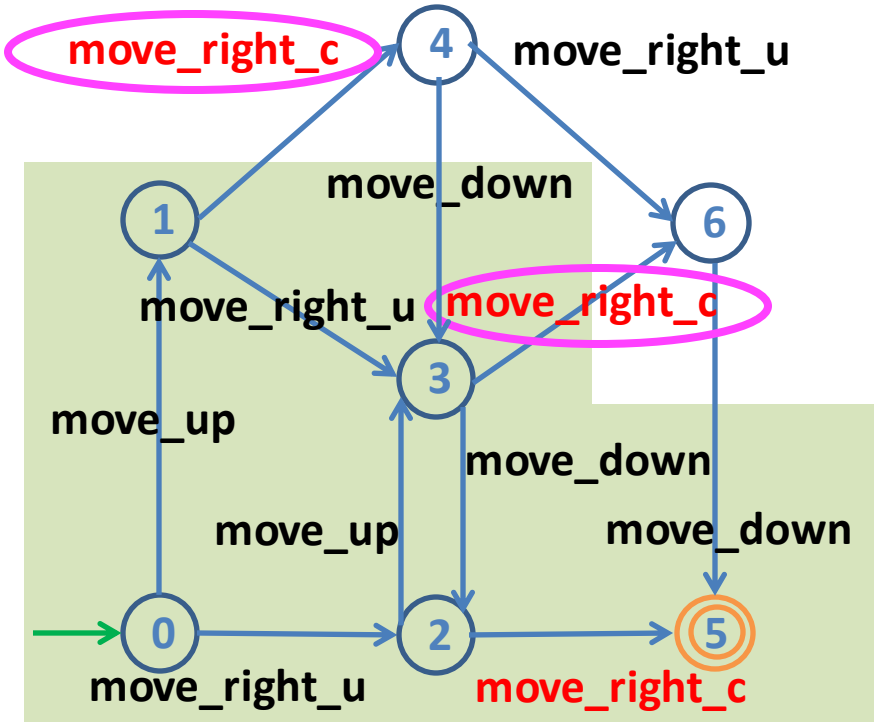
Specification **S**



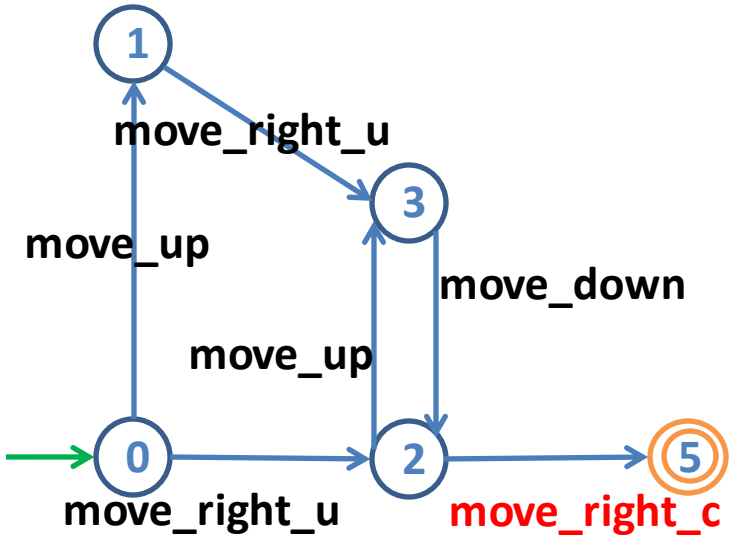
S is not controllable

A warehouse robot

ROBOT



Specification $\mathbf{S}'$

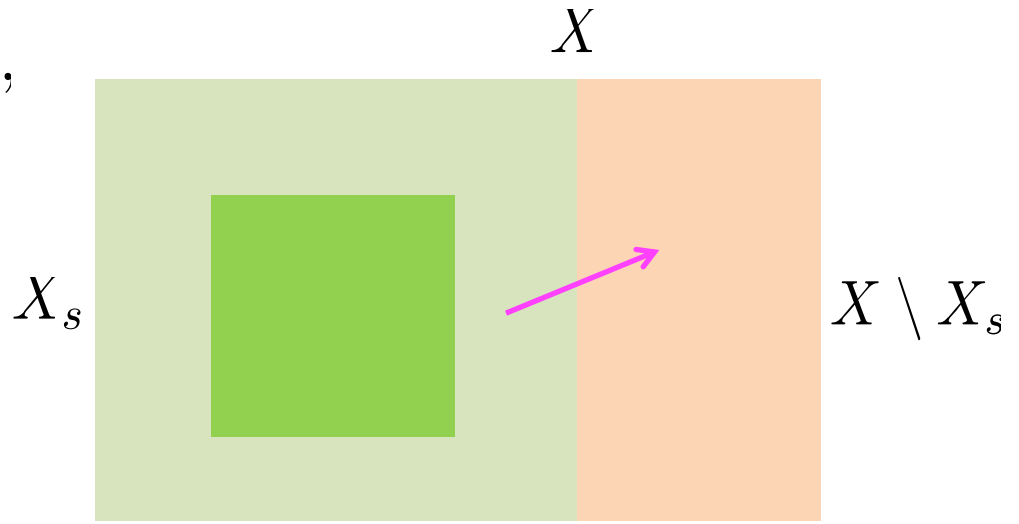


$\mathbf{S}'$ is controllable

Controllable subspecification

Plant $\mathbf{P} = (X, \Sigma, \xi, x_0, X_m)$,
$\Sigma = \Sigma_c \cup \Sigma_u$

Specification
$\mathbf{S} = (X_s, \Sigma_s, \xi_s, x_0, X_{s,m})$



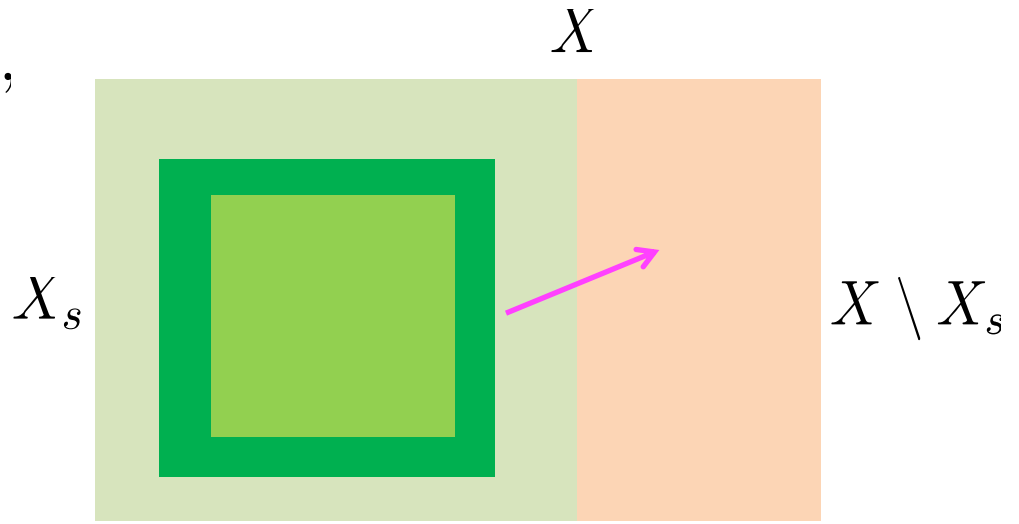
If $\mathbf{S}$ is not controllable (wrt. $\mathbf{P}$)
can we find a nonblocking subspecification $\mathbf{S}'$ *of* $\mathbf{S}$
that is controllable (wrt. $\mathbf{P}$*)?*

If so, there exists a controller $C' : X \to 2^{\Sigma_c}$ s.t. $C'/\mathbf{P} = \mathbf{S}'$

Largest controllable subspecification

Plant $\mathbf{P} = (X, \Sigma, \xi, x_0, X_m)$,
$\Sigma = \Sigma_c \cup \Sigma_u$

Specification
$\mathbf{S} = (X_s, \Sigma_s, \xi_s, x_0, X_{s,m})$

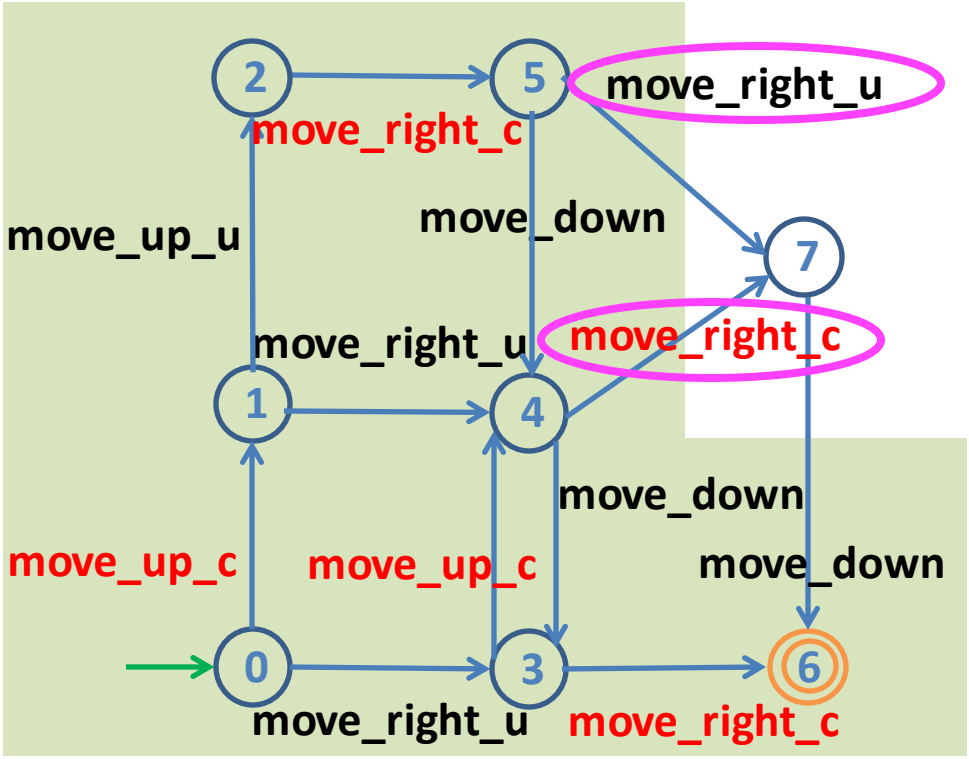


If $\mathbf{S}$ is not controllable (wrt. $\mathbf{P}$)
can we find the largest nonblocking subspecification $\mathbf{S}^*$ *of* $\mathbf{S}$
that is controllable (wrt. $\mathbf{P}$*)?*

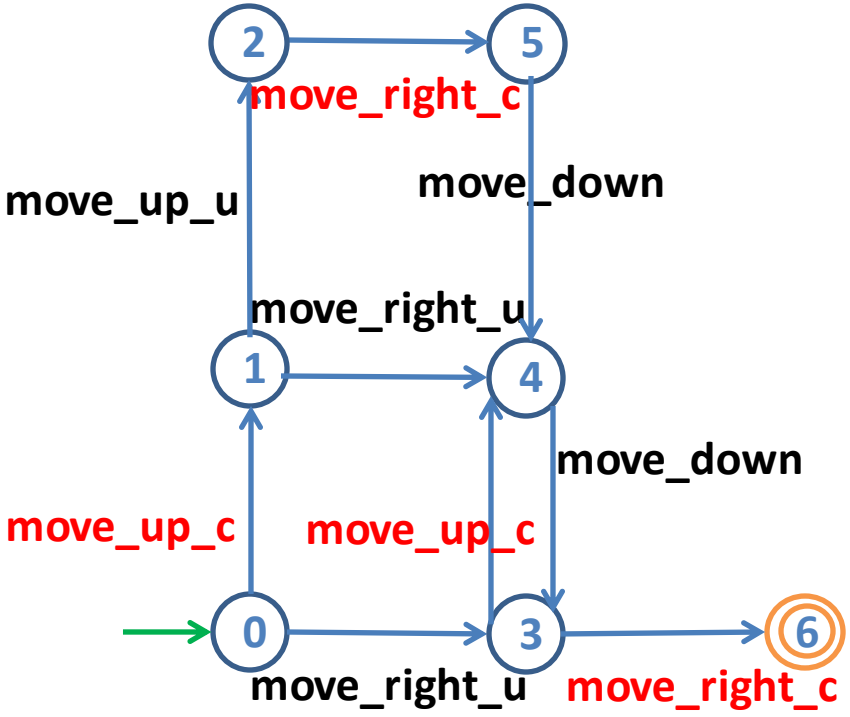
If so, the controller $C^* : X \to 2^{\Sigma_c}$ s.t. $C^*/\mathbf{P} = \mathbf{S}^*$ is optimal in that it is the maximally permissive

How to compute $\mathbf{S}^*$

ROBOT



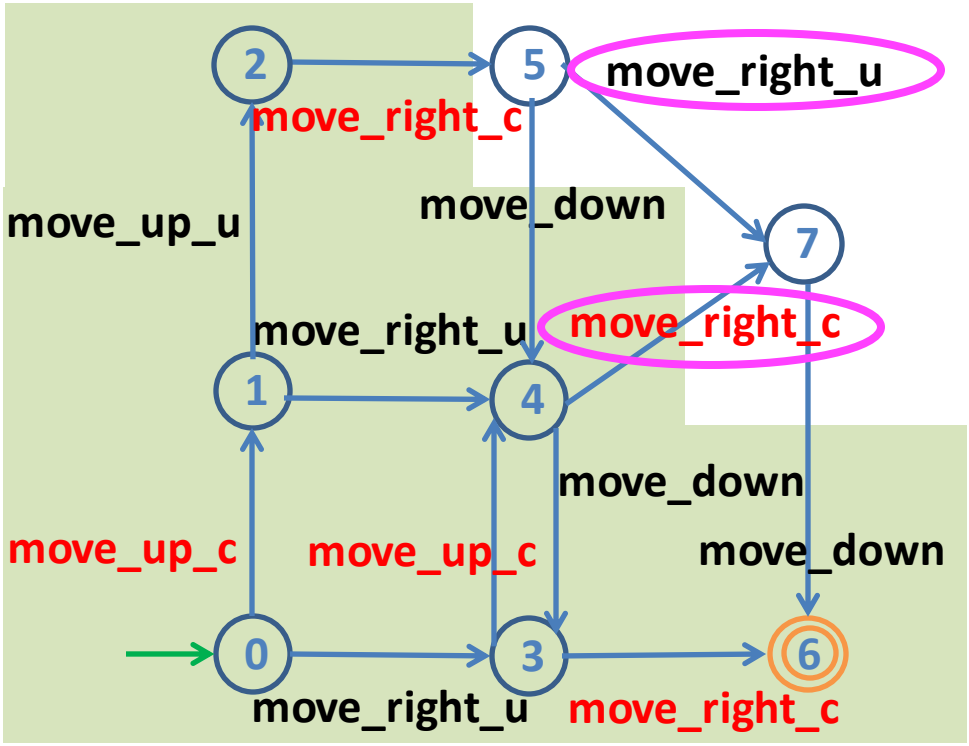
Specification $\mathbf{S}$



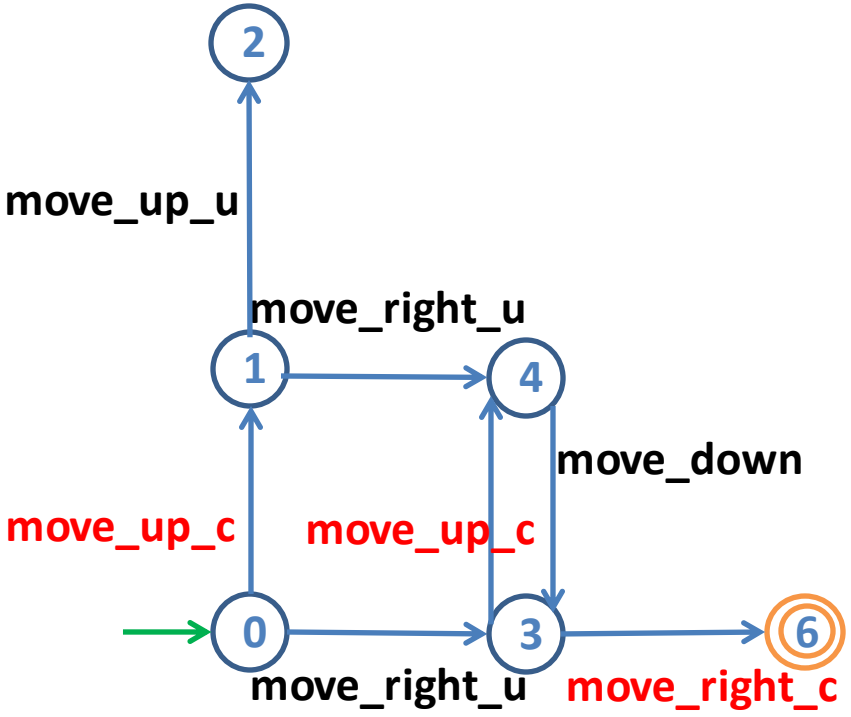
$\mathbf{S}$ is not controllable

How to compute $\mathbf{S}^*$

ROBOT



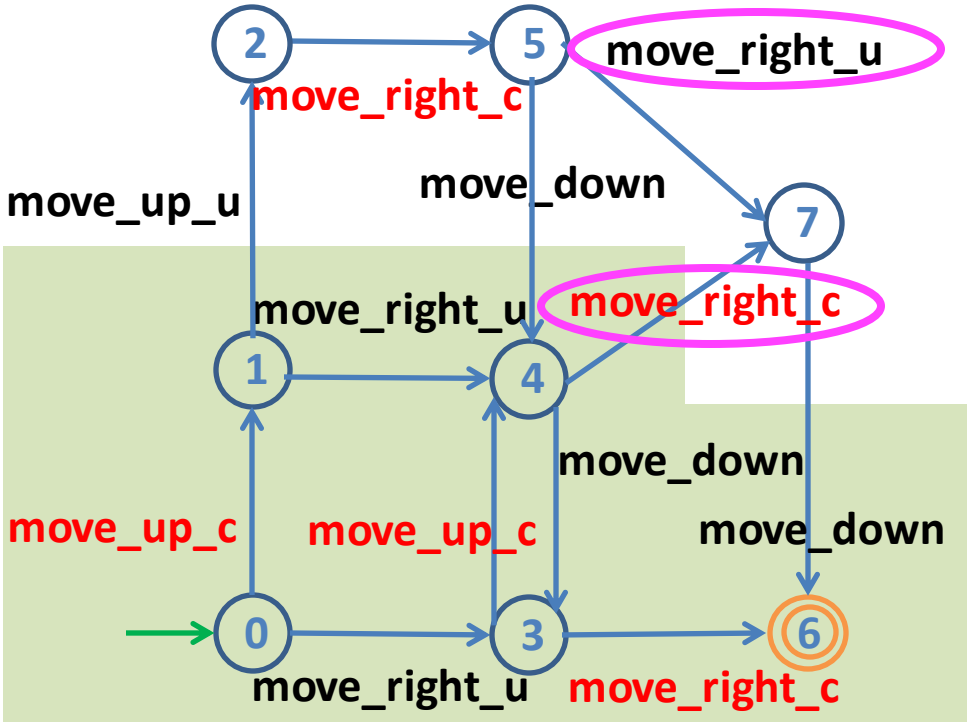
Specification $\mathbf{S}_1$



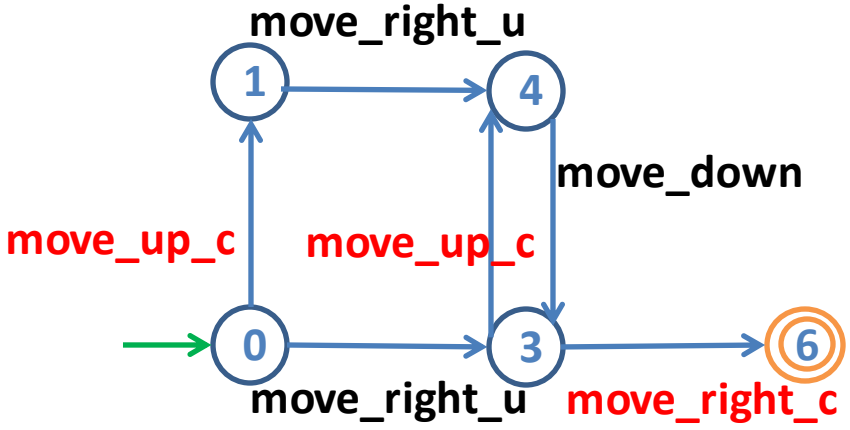
$\mathbf{S}_1$ is controllable but blocking

How to compute $\mathbf{S}^*$

ROBOT



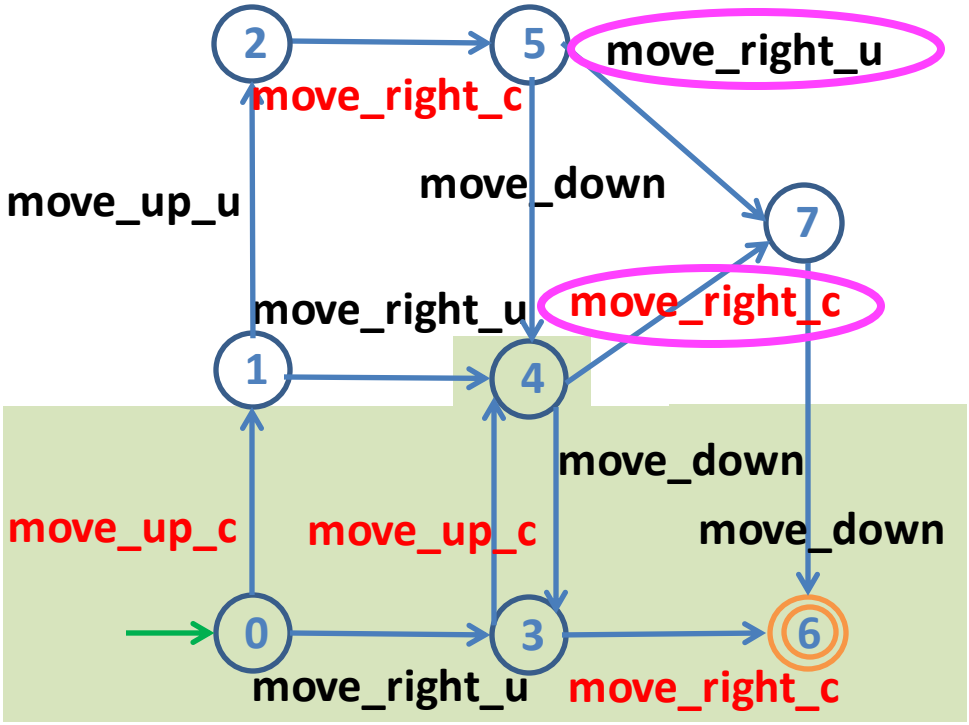
Specification $\mathbf{S}_2$



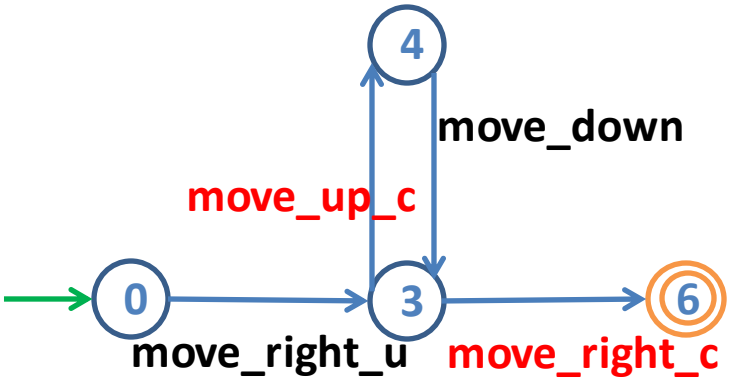
$\mathbf{S}_2$ is not controllable

How to compute $\mathbf{S}^*$

ROBOT



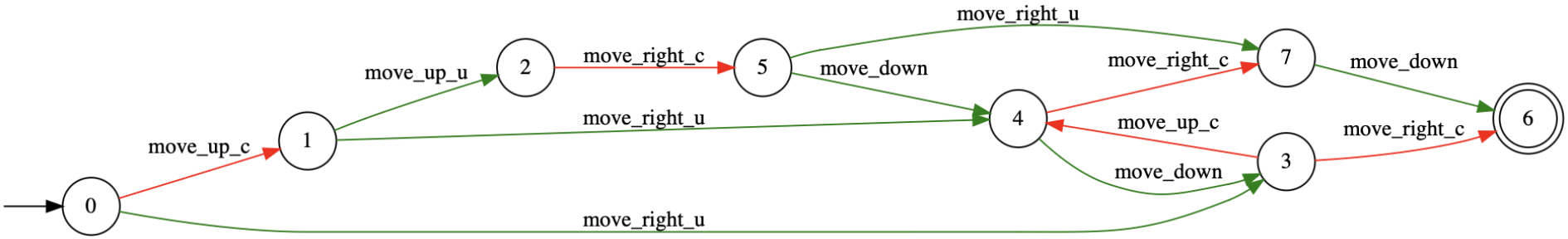
Specification $\mathbf{S}_3$



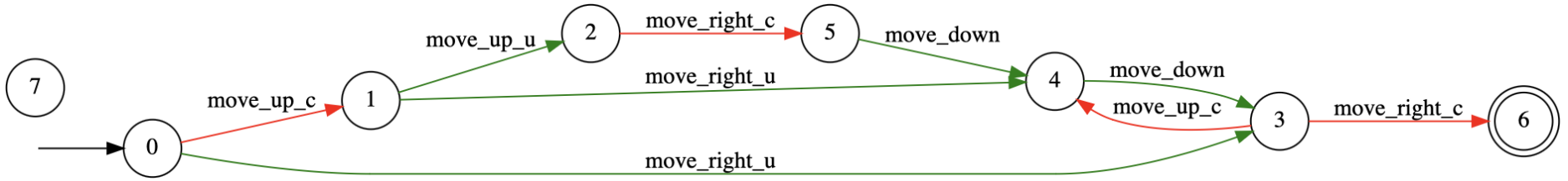
$\mathbf{S}_3$ is controllable and nonblocking

PyTCT supcon

ROBOT

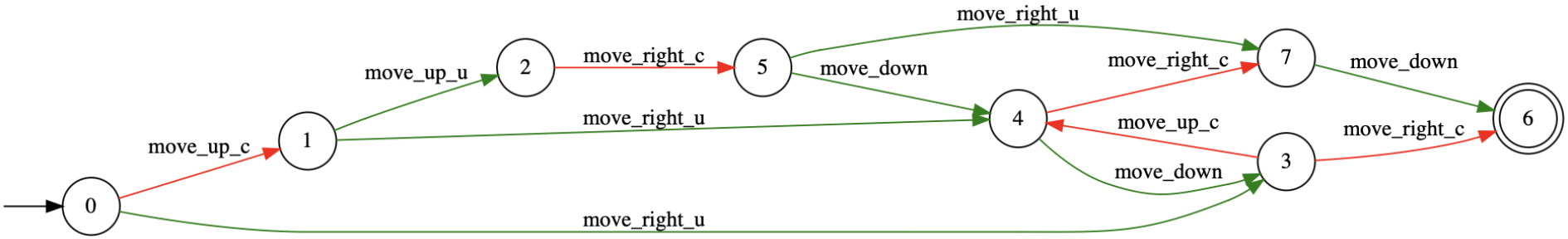


S

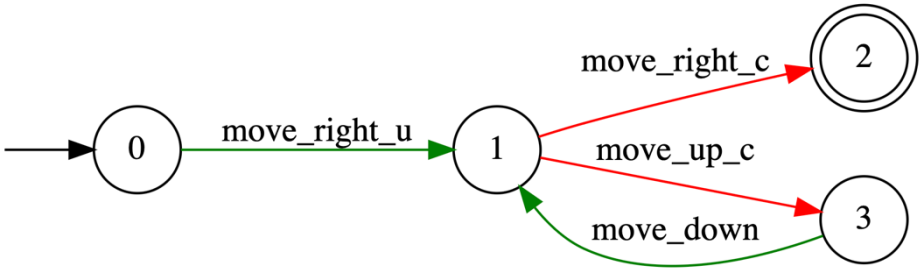


PyTCT supcon

ROBOT



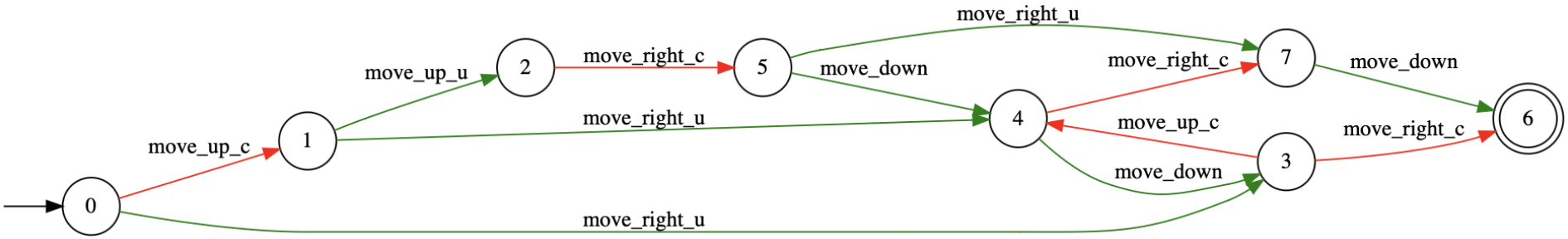
S*



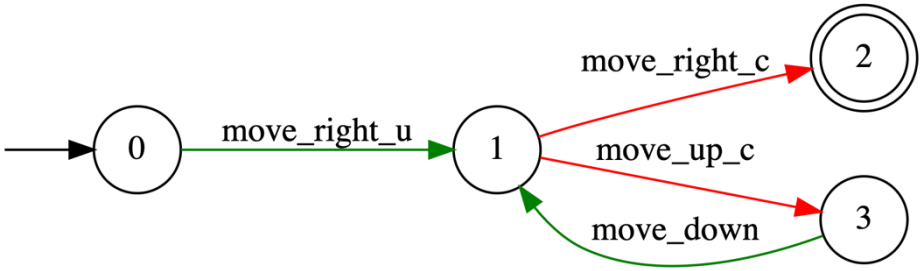

```
pytct.supcon('S_star','ROBOT','S')
pytct.display_automaton('S_star',color=True)
```

PyTCT conact

ROBOT



S*




```
ntable = pytct.conact('NROBOT','NS_star')
print(ntable)
```

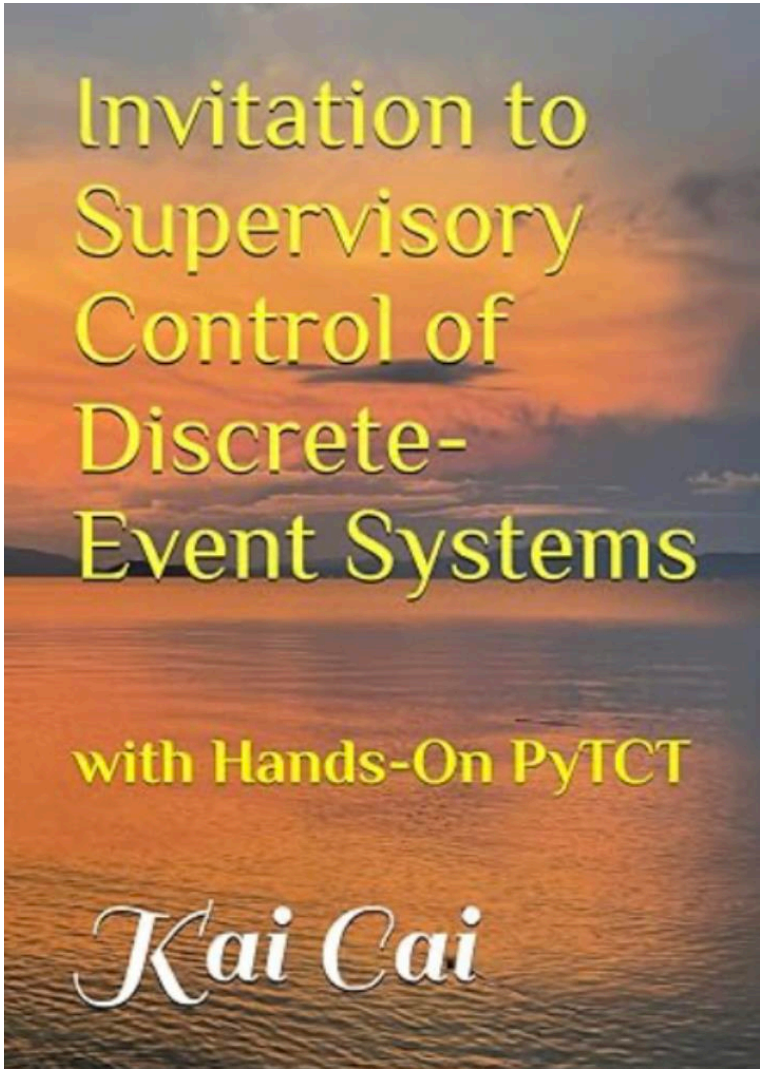
```
0: move_up_c
3: move_right_c
```

Recap

1. Create an automaton (create)
2. Properties of automaton (trim)
3. Synchronous product of automata (sync)
4. Feedback control loop (subautomaton)
5. Controllability (is_controllable)
6. Optimal supervisory control design (supcon)

DES model

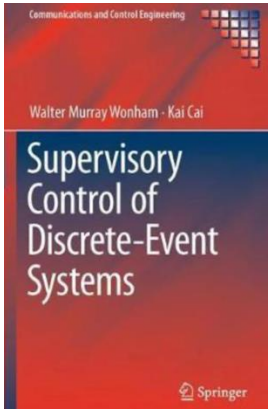
DES control

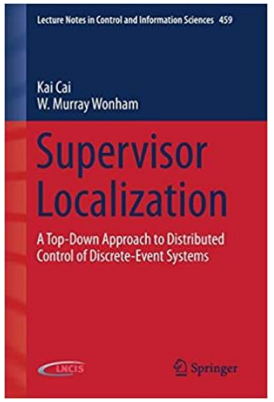


Slides and codes are available:

caikai.org/invitation-scdes

Further reading





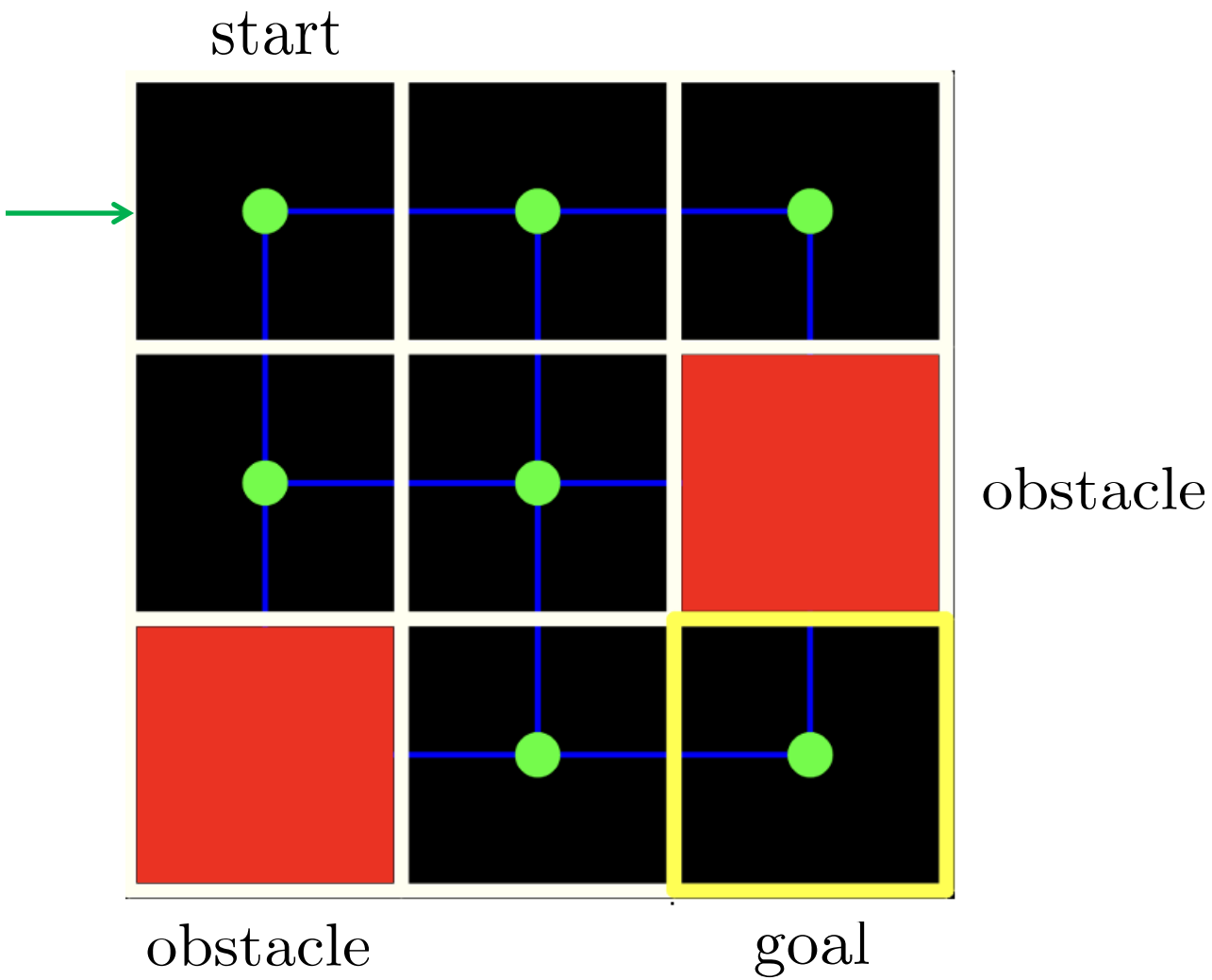
Books

- Wonham and Cai, “Supervisory Control of Discrete-Event Systems”, Springer, 2019
- Cai and Wonham, “Supervisor Localization”, Springer, 2016

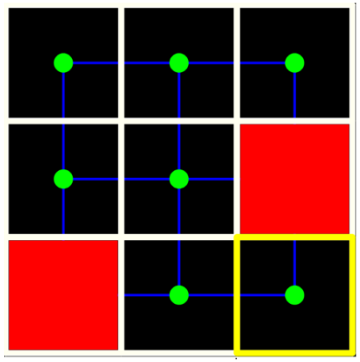
Tutorial papers

- Cai and Wonham, "Supervisory control of discrete-event systems", Encyclopedia of Systems and Control, 2nd ed., Springer, 2020.
- Cai, "Supervispr localization", Wiley Encyclopedia of Electrical and Electronics Engineering, 2019
- Wonham, Cai, and Rudie, "Supervisory control of discrete-event systems: a brief history", Annual Reviews in Control, vol. 45, pp. 250-256, 2018.

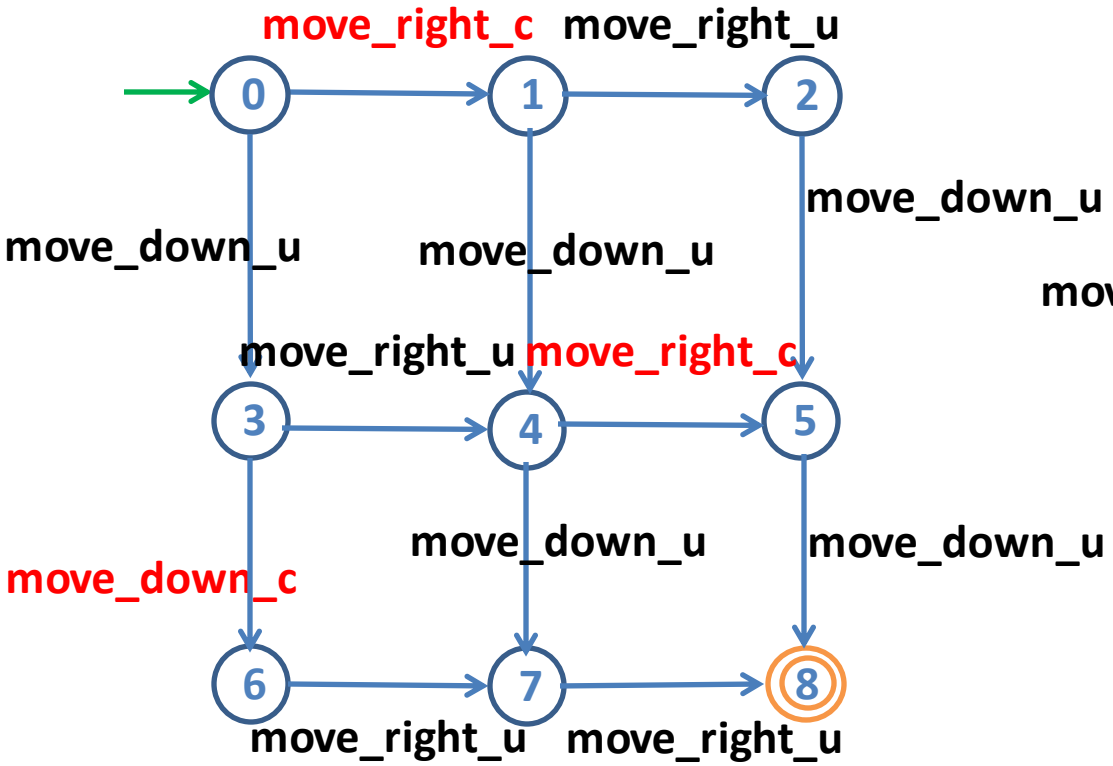
Bonus



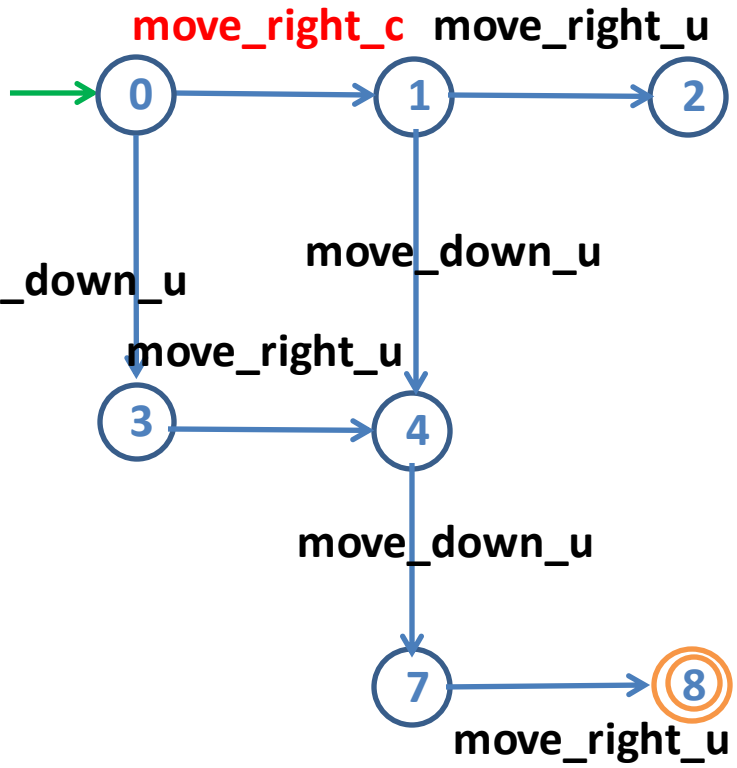
Bonus



ROBOT



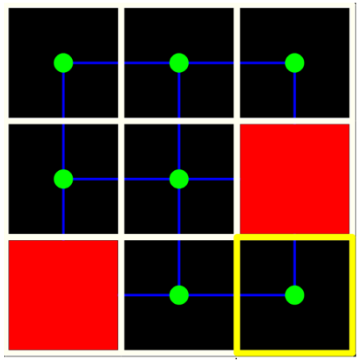
Specification **S**



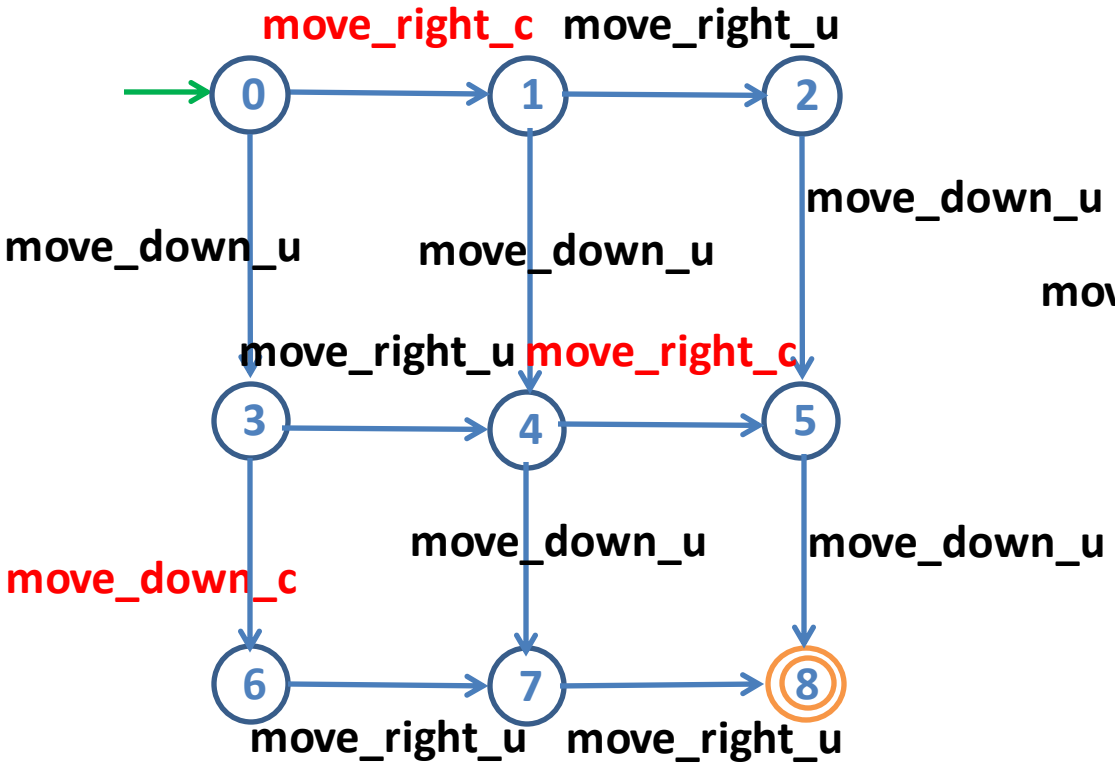
create

subautomaton

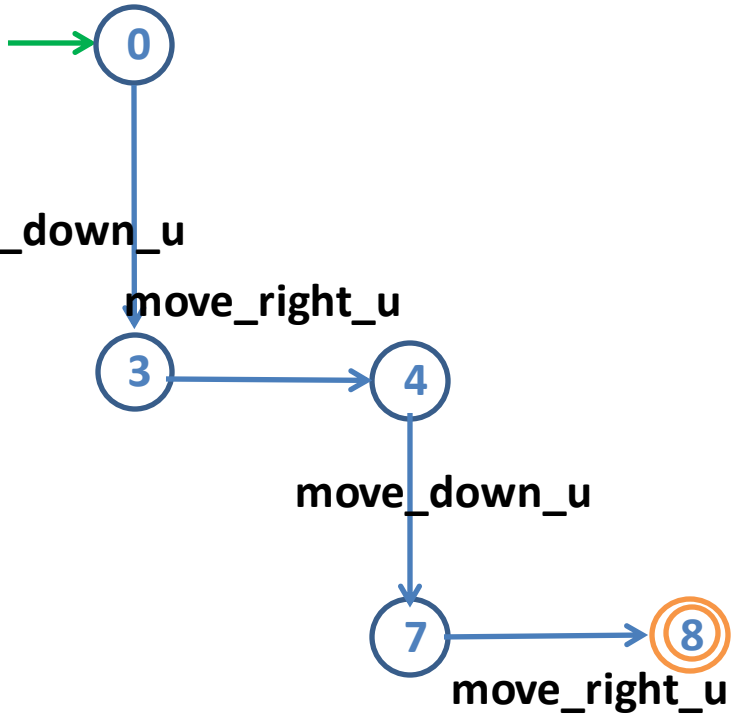
Bonus



ROBOT



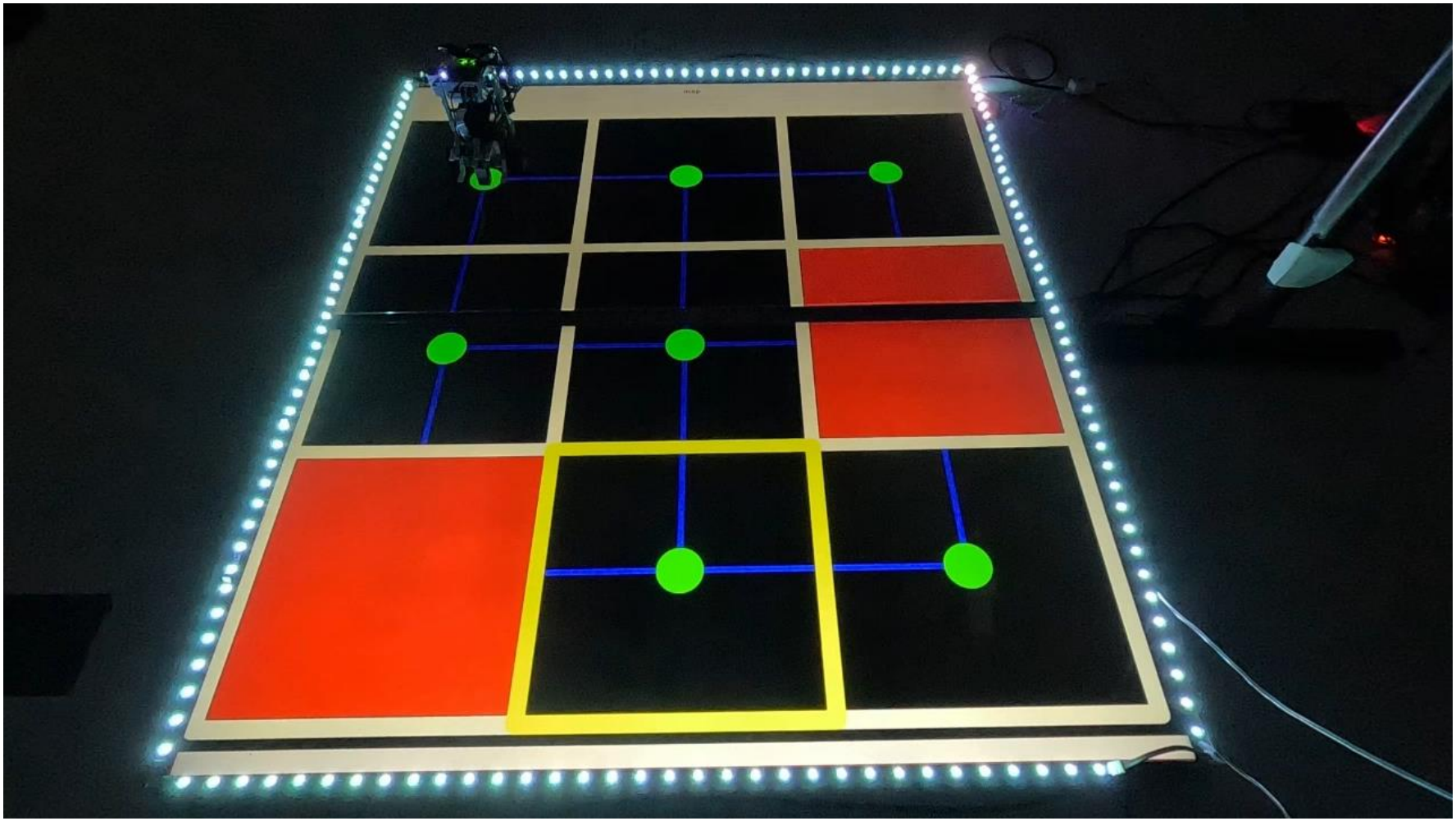
Controller $\mathbf{S}^*$



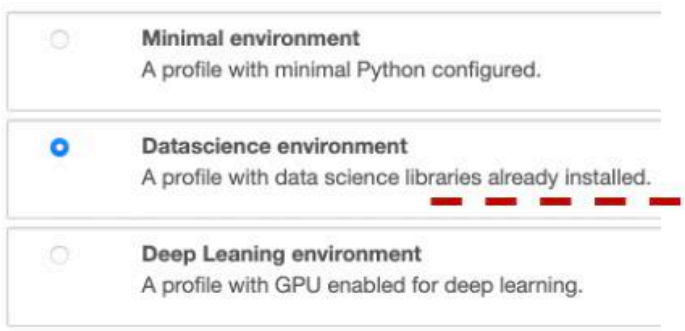
create

supcon

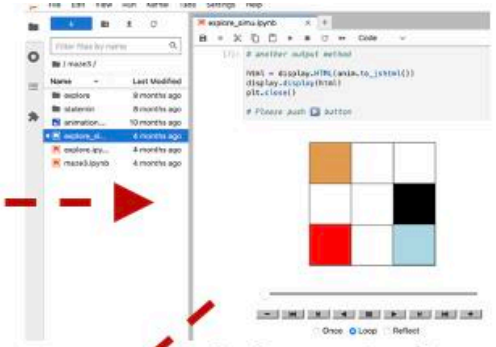
Bonus



Bonus (remote lab to be available)



jupyter.caikai.org



シミューレター

②自動で実験、動画撮影

ユーザー

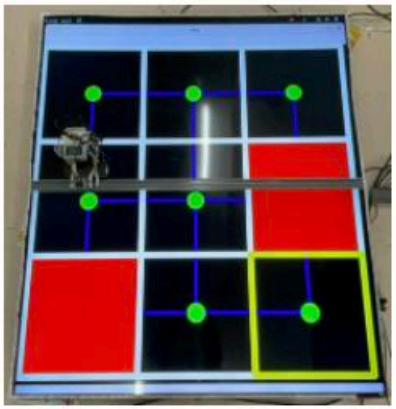
①実験の情報を送信

インターネット
(WWW)

③撮影した動画を送信

GPU
サー
バー

ロボット
実験用PC



Thank you for listening