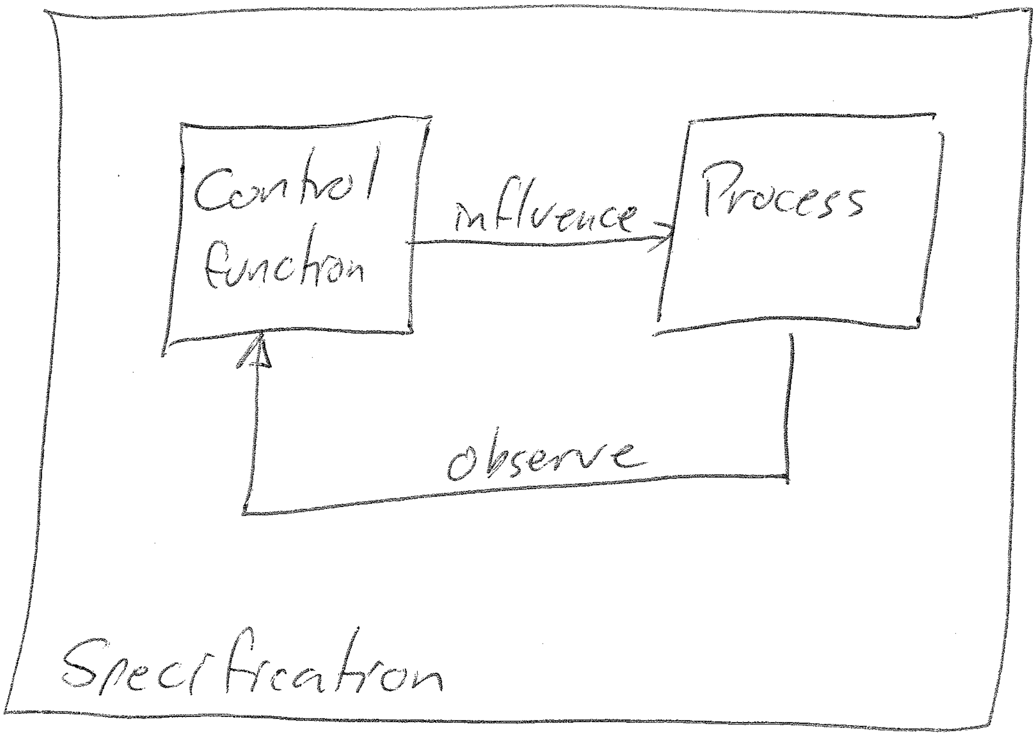
Control in general



Control loop – influence & observe

"closed loop system"

* Synthesis problem

Given process (with means to influence & control) and specification, automatically design the control function

synthesize

* Verification problem

Given process, control function and specification, automatically determine verify whether closed-loop sys fulfills the spec

Both problems solved by a

Model-based approach

Build models of the process, the control function, the specification, and the means to influence and observe

Use the models to solve the problems

Quality of the solution, depends on the accuracy of the models.

A model is an abstraction

Discrete event control

as we know it

Automata models
- Process, the plant, P or G
- Spec, Sp or K
- Control fun, supervisor, S

Synchronous composition models
- influence
- observation

The Supervisory Control Theory has developed algorithms for synthesis and verification

Automata

Directed graph

- States, the nodes, represent "situations" under which certain rules, configurations, laws, etc hold
- Transitions, the edges, represent change from one state to another

Associated with transitions are

- Events, abstract labels that are may be observed as the transition occurs

Synchronous Composition

- Composition operator, models interaction
  Same labeled events occur in two automata either simultaneously, or not at all.

Example

Stick picking game

* Two players, A and B, take alternating turns in picking sticks
  Each must take 1, 2 or 3 sticks

* Number of sticks that the players pick

Goal of the game

* The one who picks the last stick <u>loses</u>

* The one in turn when no sticks left <u>wins</u>

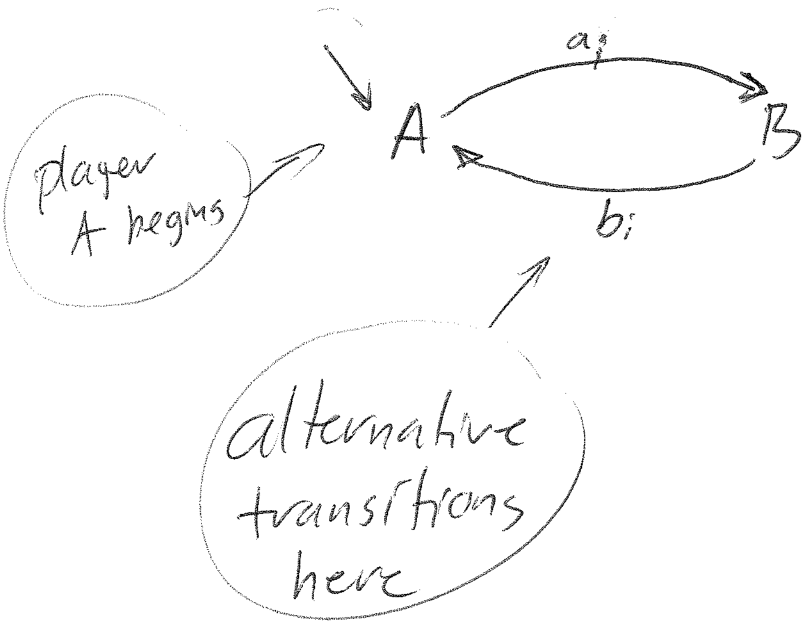
Two entities, Players and Sticks

Look at game with 7 sticks

Players

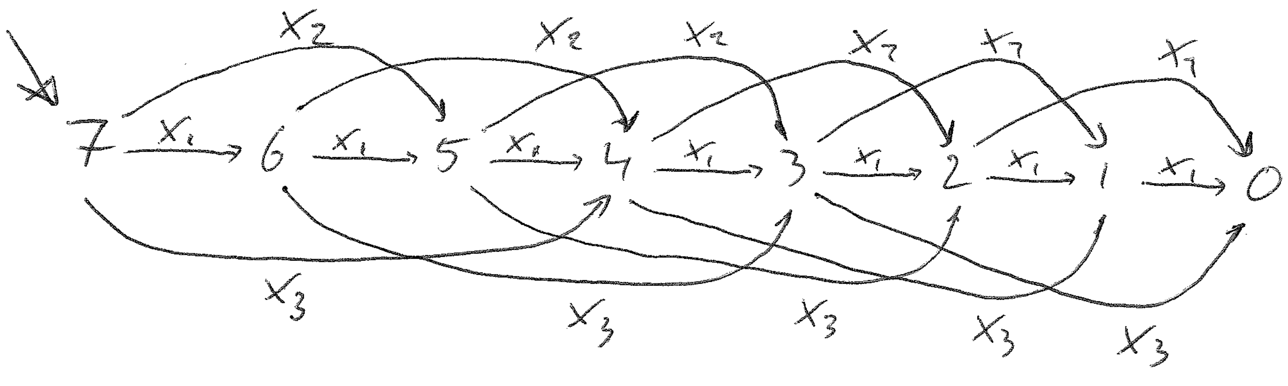
$i \in \{1, 2, 3\}$

i is the number of sticks picked



Sticks

$x \in \{a, b\}$ x is the player



Automaton, formally

Described by a tuple

$A := \langle Q_A, \Sigma_A, \delta_A, i_A \rangle$

Q_A the set of states

Σ_A the set of events, the <u>alphabet</u>

δ_A the transition function, $\delta_A : Q_A \times \Sigma_A \to Q_A$

i_A the initial state, $i_A \in Q_A$

Partial,
Not necessarily
defined for all
$\langle q, \sigma \rangle$ pairs

$Q_{Players} := \{A, B\}$

$\Sigma_{players} := \{a_1, a_2, a_3,$
$b_1, b_2, b_3\}$

$\delta_{players} := \{\langle\langle A, a_i\rangle, B\rangle,$
$\langle\langle B, b_i\rangle, A\rangle\}$

will be 6
such tuples

$i_A := A$

strings

transition function can be extended to <u>sequences of events</u>

$$\delta_A : Q_A \times \Sigma_A^* \longrightarrow Q_A$$

The empty string

Where $\Sigma_A^* := \bigcup_n \Sigma_A^n$ with $\Sigma^0 := \varepsilon$, $\Sigma^k := \Sigma^{k-1} \cdot \Sigma$

concatenation

and $\delta_A(q, \varepsilon) = q$

$$\delta_A(q, s\sigma) = \delta_A(\delta_A(q, s), \sigma)$$

Assuming $\delta_A(q, s)$ defined

Then we can define the <u>language</u> of an automaton

$$L(A) := \{ s \in \Sigma_A^* \mid \delta_A(i_A, s) \text{ defined} \}$$

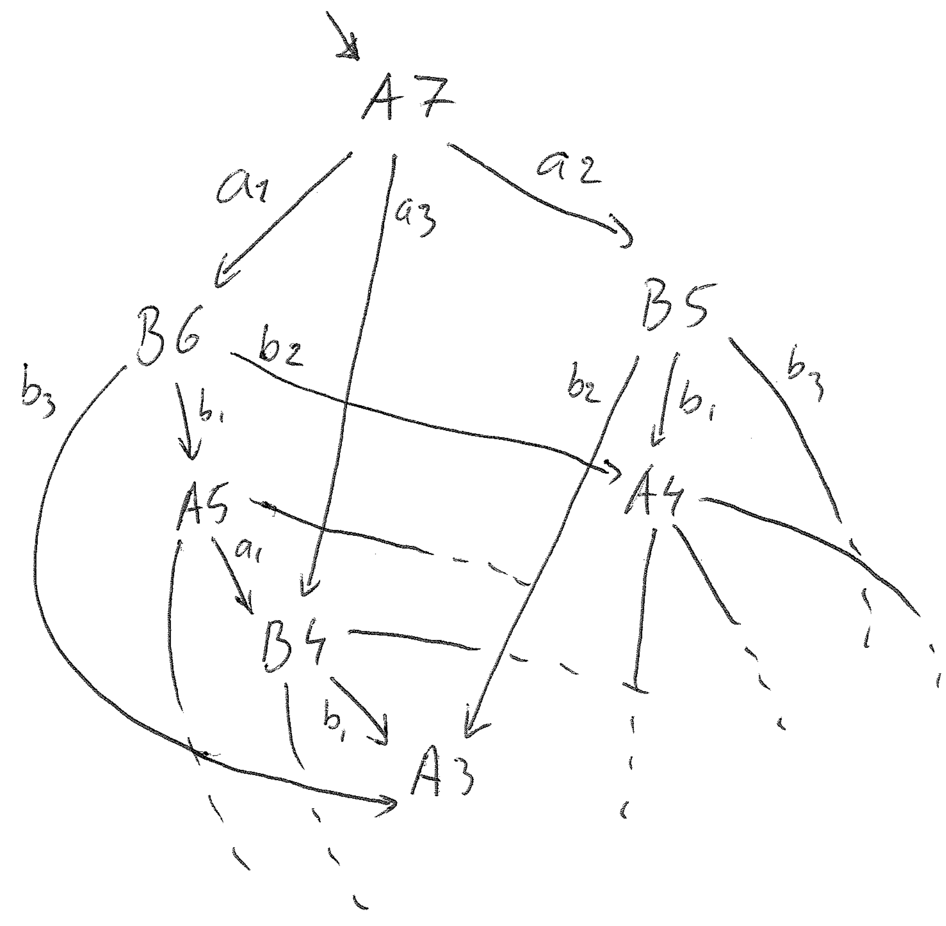
Game = interaction between players and sticks

G := Players || Sticks

synch

When player x picks i sticks, the sticks simultaneously go from state y to state y-i

in synchrony



Synch, formally

$A_1 \| A_2$ is a new automaton describing the interaction between A_1 and A_2

$$A_1 \| A_2 := \langle Q_{A_1\|A_2}, \Sigma_{A_1\|A_2}, \delta_{A_1\|A_2}, i_{A_1\|A_2} \rangle$$

where

$$Q_{A_1\|A_2} := Q_{A_1} \times Q_{A_2}$$

in practice not the full cross prod

Only reachable states are of interest

$$\Sigma_{A_1\|A_2} := \Sigma_{A_1} \cup \Sigma_{A_2}$$

$$i_{A_1\|A_2} := \langle i_{A_1}, i_{A_2} \rangle$$

$$\delta_{A_1\|A_2}(\langle q_1, q_2\rangle, \sigma) := \begin{cases} \langle \delta_{A_1}(q_1,\sigma), \delta_{A_2}(q_2,\sigma)\rangle & \sigma \in \Sigma_{A_1} \cap \Sigma_{A_2} \\ \langle \delta_{A_1}(q_1,\sigma), q_2\rangle & \sigma \in \Sigma_{A_1} \setminus \Sigma_{A_2} \\ \langle q_1, \delta_{A_2}(q_2,\sigma)\rangle & \sigma \in \Sigma_{A_2} \setminus \Sigma_{A_1} \\ \text{undefined} & \text{else} \end{cases}$$

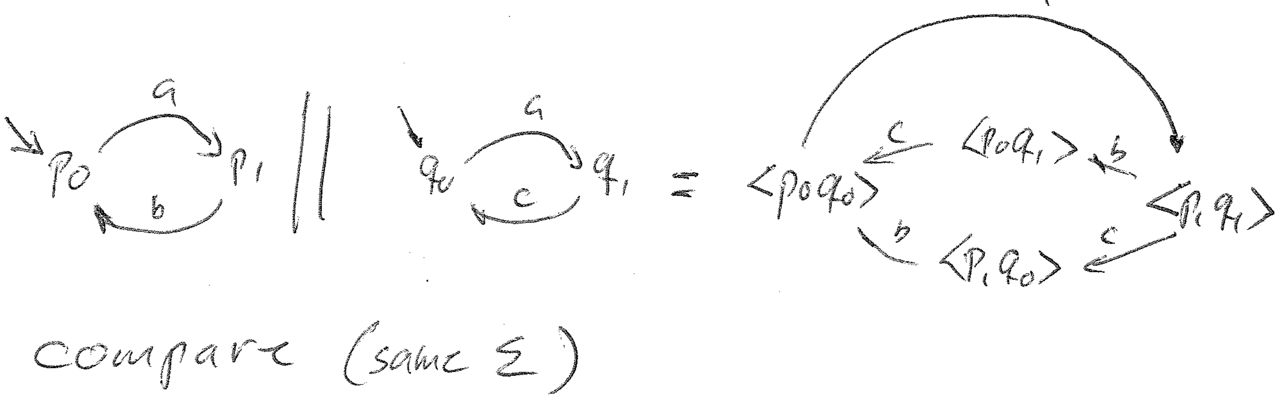
Synch, language

$$\Sigma_{A_1} = \Sigma_{A_2} \Rightarrow L(A_1 \| A_2) = L(A_1) \cap L(A_2)$$

"If we can both do exactly the same things ($\Sigma_{A_1} = \Sigma_{A_2}$) then together we only do what we both agree on ($L(A_1) \cap L(A_2)$)"

$$\Sigma_{A_1} \neq \Sigma_{A_2} \Rightarrow L(A_1 \| A_2) = L^{-1}(A_1) \cap L^{-1}(A_2)$$

"the things I can do by myself ($\Sigma_{A_i} \setminus \Sigma_{A_j}$) I will do by myself"



compare (same Σ)

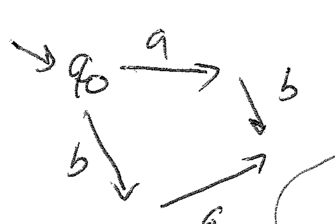
Subautomaten

An automaton $A := \langle Q_A, \Sigma_A, i_A, \delta_A, M_A \rangle$ is a <u>sub-automaton</u> of another automaton $B := \langle Q_B, \Sigma_B, i_B, \delta_B, M_B \rangle$ if

$$Q_A \subseteq Q_B$$
$$\Sigma_A = \Sigma_B$$
$$i_A = i_B$$
$$\delta_A \subseteq \delta_B$$
$$M_A \subseteq M_B$$

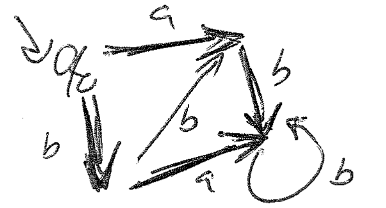
Note
This can be defined by the existence of a function of a certain structure, thus doing away with the state-space inclusion

A



$A \leq B$

B



Sub-automaton relation is

* reflexive, $A \leq A$
* transitive, $A \leq B \wedge B \leq C \Rightarrow A \leq C$
* anti-symmetric, $A \leq B \wedge B \leq A \Rightarrow A = B$

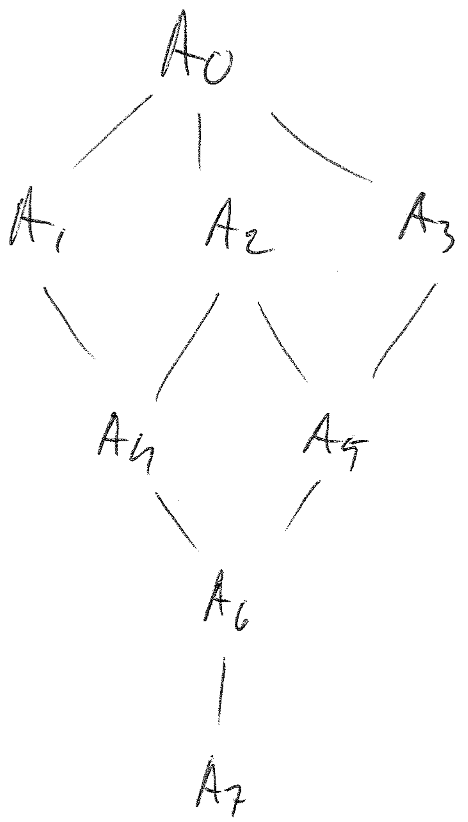
By whateve def of automata equivalence

Note also

$A \leq B \Rightarrow L(A) \subseteq L(B)$

<u>The lattice of sub-automata</u>

For an automaton A_0 all its sub-automata can be ordered into a <u>lattice</u> structure



for example
$A_4 \leq A_1$
$A_4 \leq A_2$
A_1 and A_2 unrelated

Important: For any two sub-automata, a least upper bound exists

A_4 & A_5, LUB is A_2
A_6 & A_5, LUB is A_5
A_1 & A_5, LUB is A_0

Automata refinement

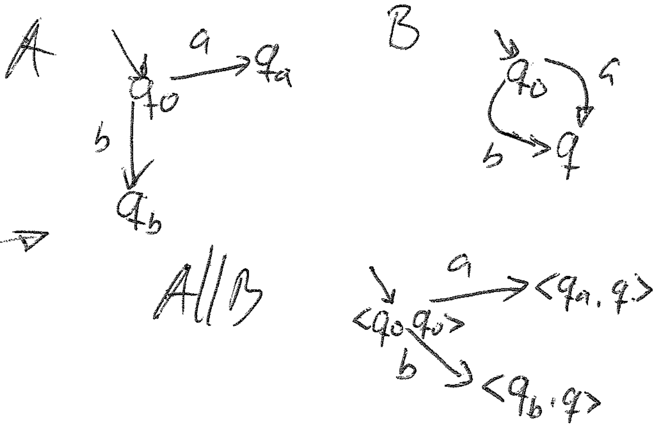
For sub-automata $A \leq B$ it holds that

$$A \| B = A$$

Is this the only way this holds?

No!

when we see that we are in q_b, we know for sure A has done b



when we see that we are in q we know B has done a or b

An automaton A <u>refines</u> automaton B when

$$A \| B = A$$

We write $A \prec B$, this relation is

* reflexive, $A \prec A$
* transitive, $A \prec B \wedge B \prec C \Rightarrow A \prec C$
* anti-symmetric, $A \prec B \wedge B \prec A \Rightarrow A = B$

Note that, $A \prec B \Rightarrow \Sigma_B \subseteq \Sigma_A$

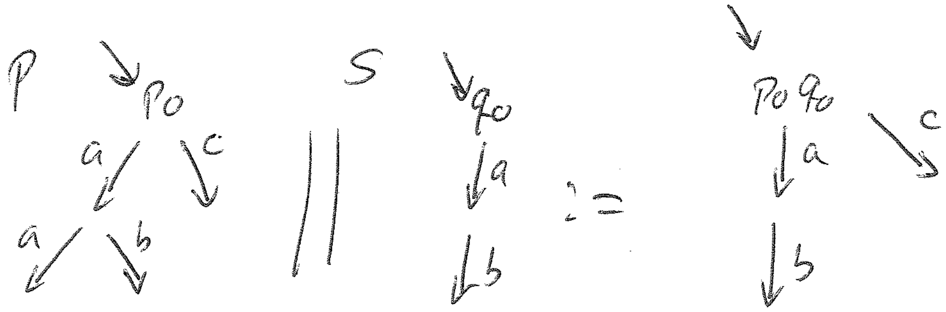
Important: $A \quad B \wedge B \prec C \Rightarrow A \prec C$

Supervisory Control

Control by S over P is modeled by synchronous composition

This is restrictive control

- influence, S does not define a transition on an event P would have transited on



Assuming $c \notin \Sigma_S$

- Observation, S observes P by following events generated by P

After event "a" above, S "knows" where P is

Even "c", S does not care about

Closed-loop system within specification

We have a plant P

Sometimes G

We have a spec Sp

Sometimes K

We want a supervisor S, such that the closed-loop system stays within the specification

$$L(P\|S) \subseteq L(P\|Sp)$$
$$L_m(P\|S) \subseteq L_m(P\|Sp)$$

What about $\Sigma_{Sp} \setminus \Sigma_P \neq \emptyset$?

Sp may be a partial spec, $\Sigma_{Sp} \subseteq \Sigma_P$
Then the total spec is $P\|Sp$

Sp may be a static spec, only marking (or forbidding) states in the plant. Then
$Sp = P'$ and $P\|Sp = P\|P' = P'$

Assuming no plant states marked (and none forbidden)

Specification – A is to win

This is a static spec that can be introduced simply by marking and/or forbidding states in G

Marked state A0

A's turn to pick
No sticks left

Task: Synthesize supervisor that guarantees that A wins

Should guarantee A0 state always reachable

Marked states

Some states are "significant"
sub-tasks of special importance

We include the notion of
<u>marked states</u>

Not necessarily final!

that represent these "significant" states

Augment the automaton description

$A := \langle Q_A, \Sigma_A, \delta_A, i_A, M_A \rangle$ where

$M_A \subseteq Q_A$ is the set of marked states

Marked language

$L_m(A) := \{ s \in L(A) \mid \delta_A(i_A, s) \in M_A \}$

Marked states for composition

$M_{A_1 \| A_2} := M_{A_1} \times M_{A_2}$

Nonblocking

Supervisor should always be non-blocking

At least one marked state always reachable

$$L(P\|S) \subseteq \overline{L_m(P\|S)}$$

Here $L_m(P\|S)$ are the strings that reach marked states

$\overline{L_m(P\|S)}$ strings extentable to reach marked states

$L(P\|S)$ is all possible strings

"All allowed strings should be extendable to reach marked states"

It always holds that $L_m(P\|S) \subseteq \overline{L_m(P\|S)} \subseteq L(P\|S)$

Formally:

$$\overline{L_m(P\|S)} := \{ s \in \Sigma_{P\|S}^* \mid \exists t \in \Sigma_{P\|S}^* : st \in L_m(P\|S) \}$$

Controllability

For the stick-picking game, we want a supervisor that describes how A is to play to be guaranteed to win.

Must take into account that, from A's perspective, we have no control over B's picks.

B's events are uncontrollable

A's events are controllable

$$\Sigma := \Sigma_c \cup \Sigma_u , \quad \Sigma_c \cap \Sigma_u = \emptyset$$

Controllable events

uncontrollable events

Supervisor controllability

Supervisor must be such that it never influences (restricts) an uncontrollable event.

"uc"

In each state, S must define transitions on the uc-events P defines in its corresponding state

$\forall s \in L(P\|S) \quad \langle p,q \rangle := \delta_{P\|S}(i_{P\|S}, s) \quad \Sigma(q) \cap \Sigma_u \subseteq \Sigma(p)$

where $\Sigma(q)$ is the set of events defined from q

In language terms

$$L(P\|S)\,\Sigma_u \cap L(P) \subseteq L(P\|S)$$

or equivalently

$$L(S)\,\Sigma_u \cap L(P) \subseteq L(S)$$

why are these equivalent?

These are equivalent, because...

(Assume equal alphabets, $\Sigma_P = \Sigma_S$, no loss of generality but gain of clarity)

$$L(P\|S)\Sigma_u \cap L(P) \subseteq L(P\|S)$$

$\iff$ by def of synch

$$[L(P) \cap L(S)]\Sigma_u \cap L(P) \subseteq L(P) \cap L(S)$$

$\iff$ since all "strings" of Σ_u are the same length compare Beta

$$L(P)\Sigma_u \cap L(S)\Sigma_u \cap L(P) \subseteq L(P) \cap L(S)$$

$\iff$ ordinary set algebra $A \cap B \subseteq B \cap C \iff A \cap B \subseteq C$

$$L(P)\Sigma_u \cap L(S)\Sigma_u \cap L(P) \subseteq L(S)$$

$\iff$ simple rewrite

$$[L(S)\Sigma_u \cap L(P)] \cap [L(P)\Sigma_u \cap L(P)] \subseteq L(S)$$

$\iff$ write out the definitions

$$\{s\sigma \in L(P) \mid s \in L(S) \wedge \sigma \in \Sigma_u \wedge s \in L(P)\} \cap \{s\sigma \in L(P) \mid s \in L(P) \wedge \sigma \in \Sigma_u\} \subseteq L(S)$$

$\iff$ obviously, $L(S)\Sigma_u \cap L(P) \subseteq L(P)\Sigma_u \cap L(P)$

$$L(S)\Sigma_u \cap L(P) \subseteq L(S)$$

Minimally restrictive supervisor

We have that the supervisor should

* Keep closed-loop system within the spec

$$L(P\|S) \subseteq L(P\|K)$$

$$L_m(P\|S) \subseteq L_m(P\|K)$$

* Be nonblocking

$$L(P\|S) \subseteq \overline{L_m(P\|S)}$$

* Be controllable

$$L(P\|S)\Sigma_u \cap L(P) \subseteq L(P\|S)$$

Is there a known supervisor that fulfills this?

Yes, there is!

The <u>null</u> supervisor, $\emptyset_{\Sigma_P} := \{\emptyset, \Sigma_P, \emptyset, \emptyset, \emptyset\}$

Guarantees that nothing goes wrong, by guaranteeing that nothing goes on!

Not very useful

Additional requirement . . .

Supervisor should be <u>minimally restrictive</u>

equivalently "maximally permissive"

Should control, that is <u>restrict</u>, as little as necessary.

Should allow the plant the largest possible amount of freedom, within the other constraints

<u>Known facts</u>, given P and K

A unique minimally restrictive, controllable S does exist and is calculatable

A unique minimally restrictive, non-blocking S does exist and is calculatable

A unique minimally restrictive, controllable and non-blocking S does exist and is calculatable

Proof of existence

Theorem: There exists a controllable supervisor S such that $P \| S \leq S_p$ if and only if there exists a controllable sub-automaton $S' \leq S_p$ such that $S' \prec P$

Proof:

($\Leftarrow$) Assume S' exists, let $S = S'$, then since $S' \prec P$, $P \| S' = S' \leq S_p$.
Since S' is controllable, so is S.

($\Rightarrow$) Assume S exists, let $S' = P \| S$, then since $P \| S \leq S_p$, it holds that $S' \leq S_p$
Clearly $P \| S \prec P$, and since S is controllable, so is $P \| S$ and hence S'

This tells us that if we can start with S_p and find within it S' such that $S' \prec P$, then S' is S.

Then we also know that $P \| S = S$, so that we can give the closed loop system any property that we give to S, such as non-blocking and minimally restrictive

Monolithic synthesis algorithm

Given, P and S_P (as automata)
Return supervisor S (automaton)

1. $S_0 := f(P \| S_P)$ ← Synch here guarantees $S_n \curlyvee P$
   $n := 1$
2. $S_n := \text{Controllable}(S_{n-1})$
3. $S_{n+1} := \text{Nonblock}(S_n)$
4. If $S_n = S_{n+1}$ then goto 5
   else $n := n+2$, goto 2
5. Return $S := S_n$

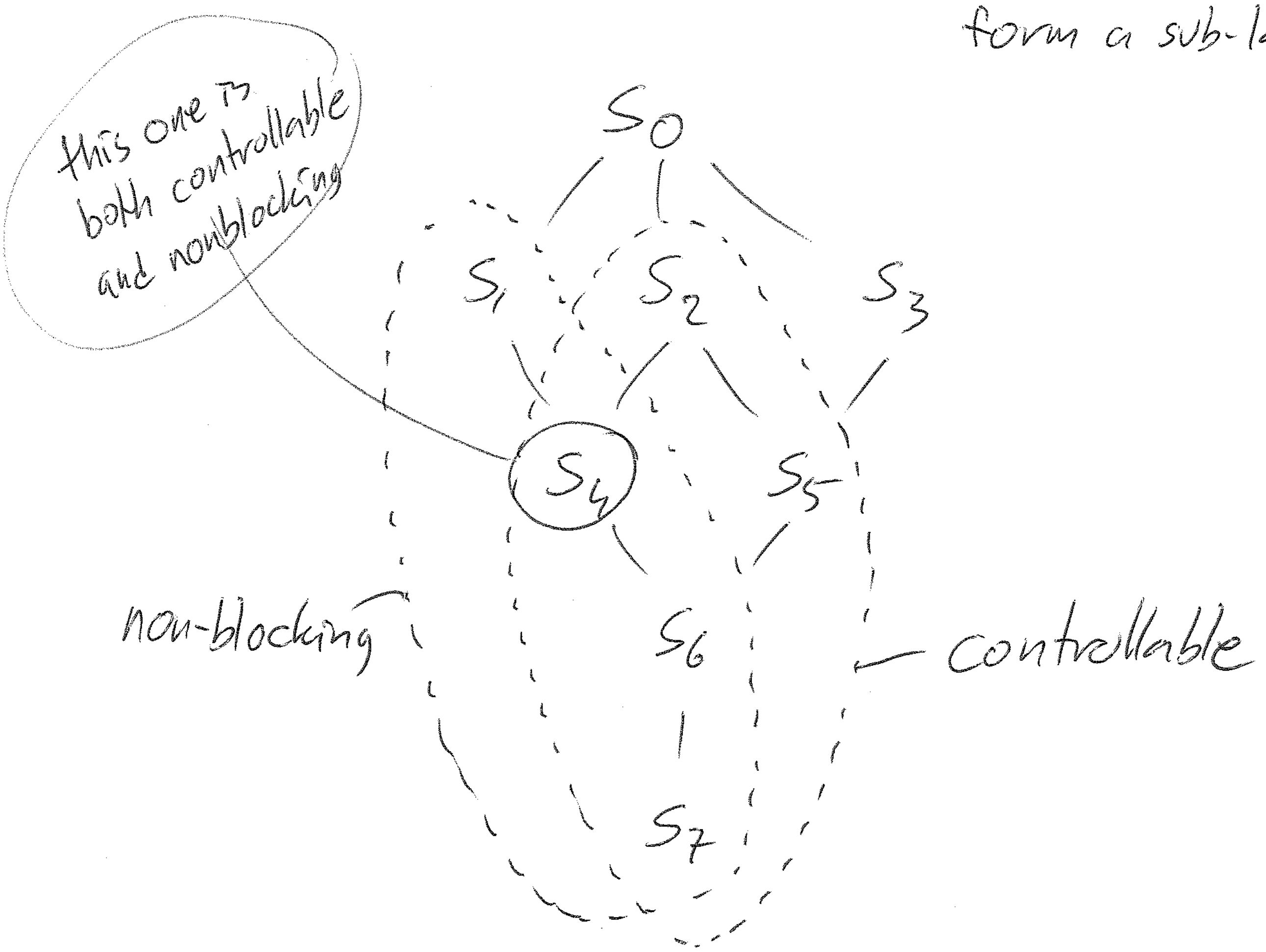
Here, $f(P \| S_P)$ finds uncontrollable <u>states</u> during synchronous composition.

Controllable (S_{n-1}) calculates the largest controllable sub-automaton of S_{n-1}

Nonblocking (S_n) calculates the largest non-blocking sub-automaton of S_n

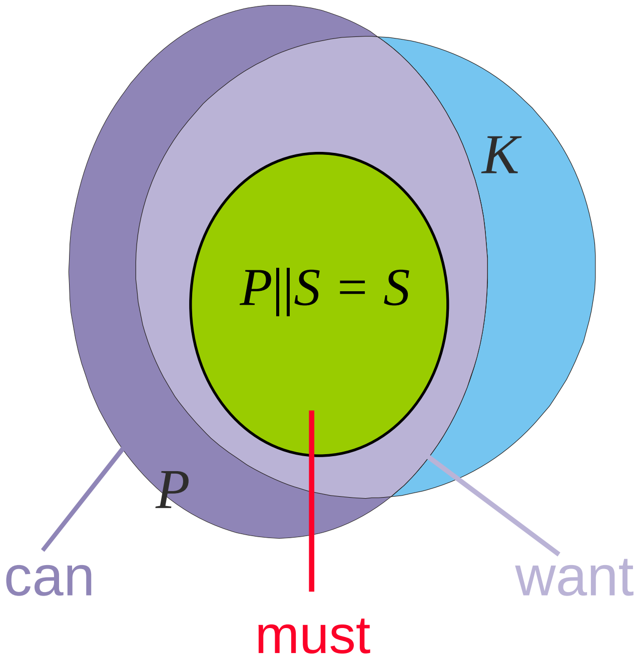
Monolithic synthesis algo

All controllable sub-automata form a sub-lattice
All non-blocking sub-automata form a sub-lattice
The intersection of two sub-lattice form a sub-lattice



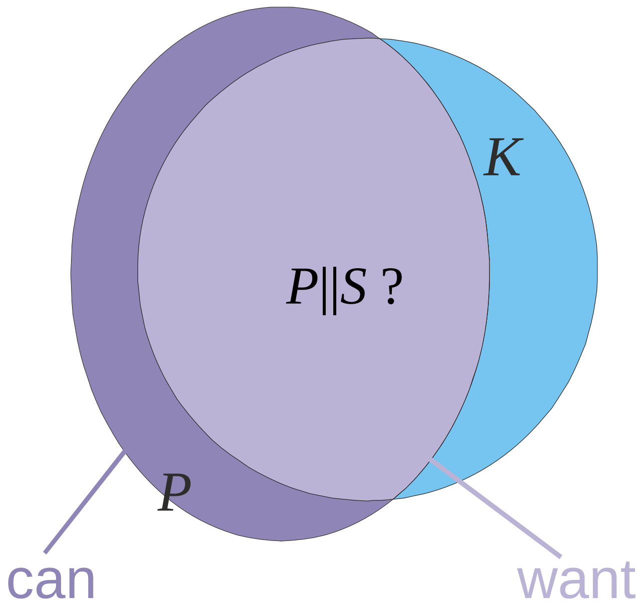
The algorithm iterates between the set of controllable and the set of non-blocking sub-automata, until it finds an automaton that is both controllable and non-blocking

Supervisor - Task



- Given
  - Process model, *P*
  - Specification, *K*
- Calculate supervisor *S*
  - Within the spec $L(P\|S) \subseteq L(P\|K)$
  - Non-blocking $L(P\|S) \subseteq \overline{L_m(P\|S)}$
  - Controllable $L(P\|S)\Sigma_u \cap L(P) \subseteq L(P\|S)$
  - Max permissive $P\|S' \leq P\|S$
- Problem
  - Blocking
  - Un-controllable events

Supervisor - Verification



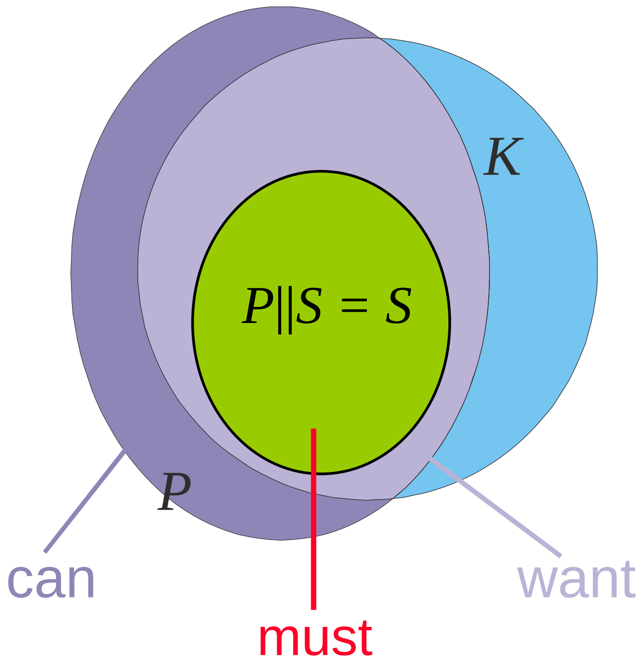
$$L(P\|S) \subseteq L(P\|K)$$

$$L(P\|S) \subseteq \overline{L_m(P\|S)}$$

$$L(P\|S)\Sigma_u \cap L(P) \subseteq L(P\|S)$$

- Given *P*, *S* and *K,* **verify** that
  - *S* ”works” properly
- *S* ”works”
  - Controllable
  - Nonblocking
- *P*||*S* fulfills the specification
  - Undesired states are avoided
  - Undesired strings avoided
  - Language inclusion

Supervisor - Synthesis



$$L(P\|S) \subseteq L(P\|K)$$

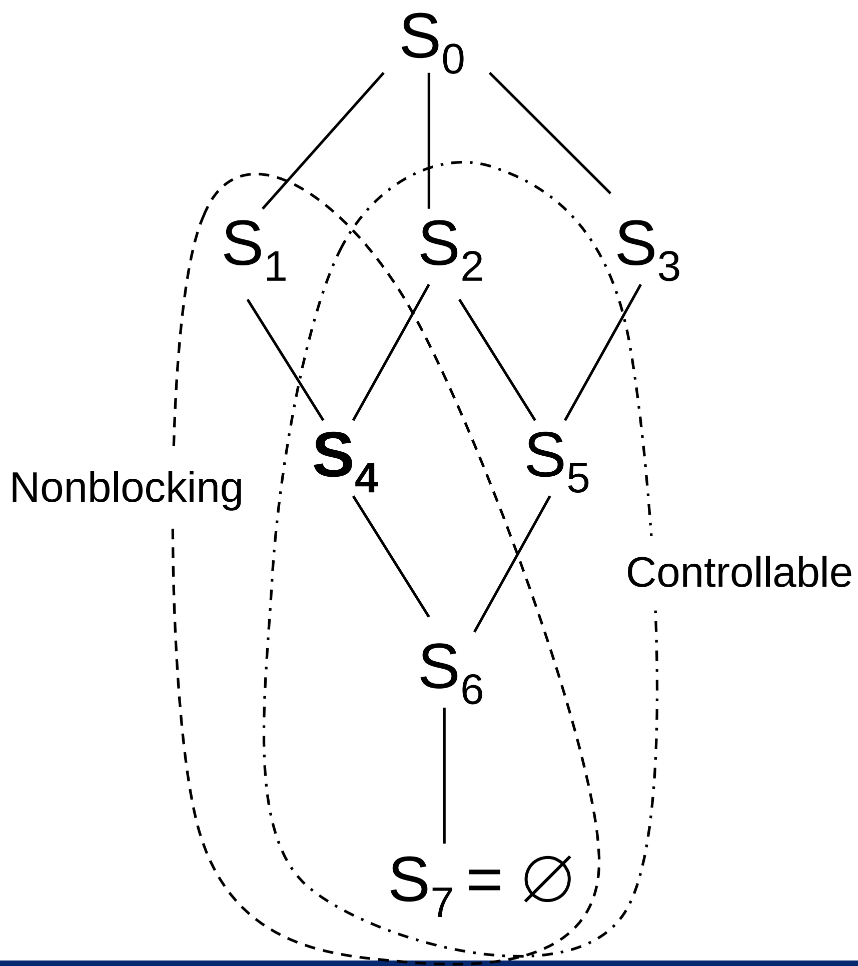
$$L(P\|S) \subseteq \overline{L_m(P\|S)}$$

$$L(S)\Sigma_u \cap L(P) \subseteq L(S)$$

- **Iterative** calculation, $S_0 = P||K$
  - Forbid undesired states
    - If uncontrollable, make controllable, S_i
    - If blocking, make nonblocking, S_{i+1}
    - Etc...
  - Terminates at **fixpoint**, $S_i = S_{i+1}$
- Optimality, $P||S = S \leq S_0$
  - A **unique largest** supervisor always exists
  - Maximally permissive, minimally restrictive
  - Allows P maximal freedom within the spec

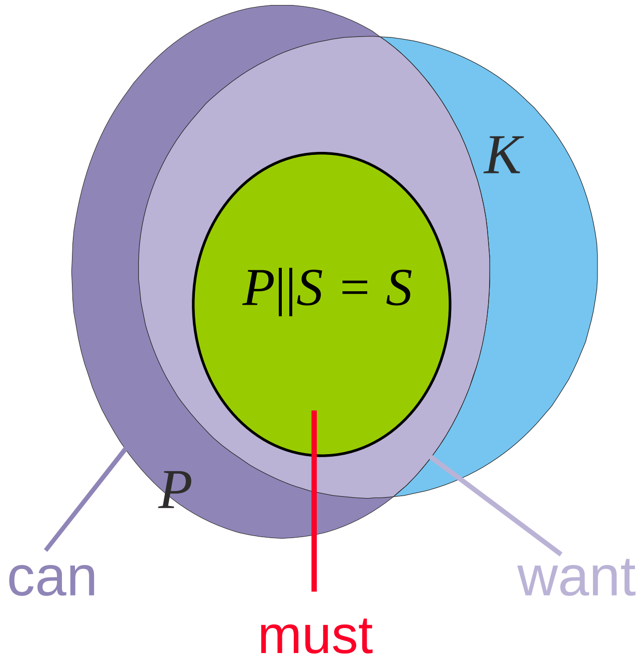
Synthesis can be viewed as a series of verification tasks

Supervisor – Minimally Retrictive



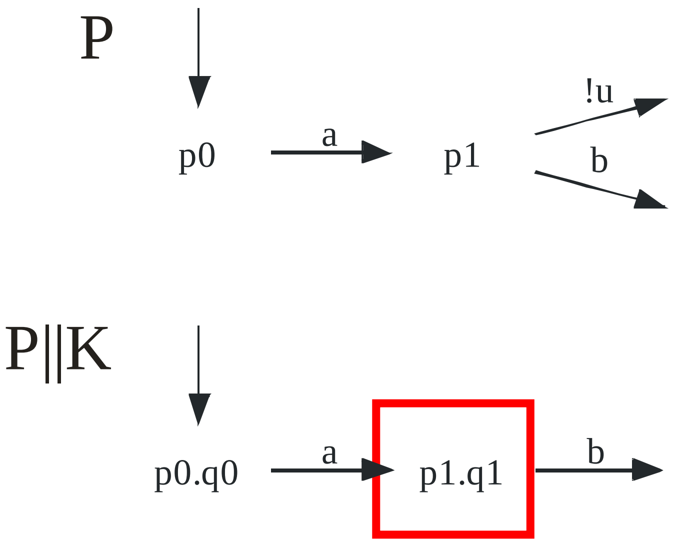
- Calculates sub-automata
  - Can be ordered in a structure
  - **Lattice**
- Unique element exist
  - Unique *largest* element, S0
  - Unique *smallest* element, 0-automaton
- Set of all **controllable** sub-automata
  - Has unique largest element, S2
- Set of all **non-blocking** sub-automata
  - Has unique largest element, S1
- Intersection controllable and non-blocking
  - Unique largest solution, S4

Supervisor - Synthesis



- Algorithm
  1. Calculate $T_0 = P||K$
  2. Find un-controllable states $S_0 = f(P, T_0)$
  3. $S_{i+1} = SupNB(S_i)$
  4. $S_{i+2} = SupC(S_{i+1})$
  5. If $S_{i+2} \neq S_{i+1}$, go to 3
  6. $S := S_{i+1}$

Supervisor – Finding Un-controllable States

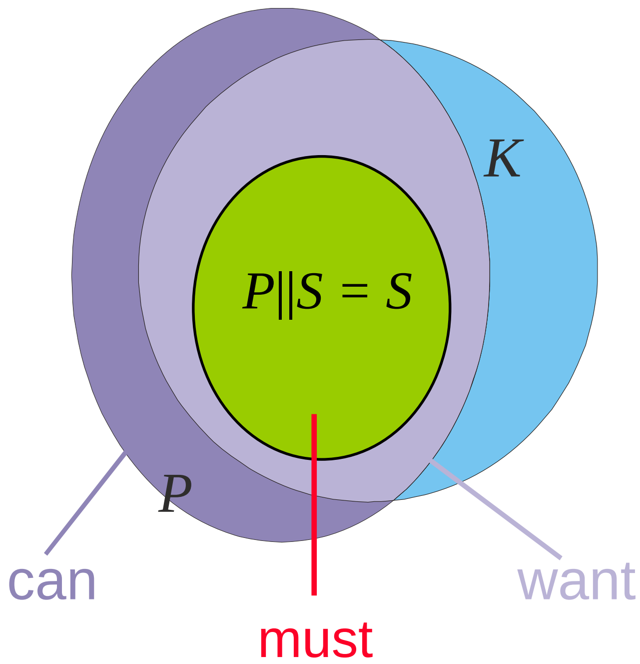


- Synch P||K
  - Compare P||K with P
  - If exists uc-event from state p
  - Not exist from state <p,q>
  - Then <p,q> **un-controlable** *state*
- Can be done while synching
  - If uc-event disappears
  - Mark state as un-controllable
  - State is forbidden

$$\forall s \in L(P \| K)$$

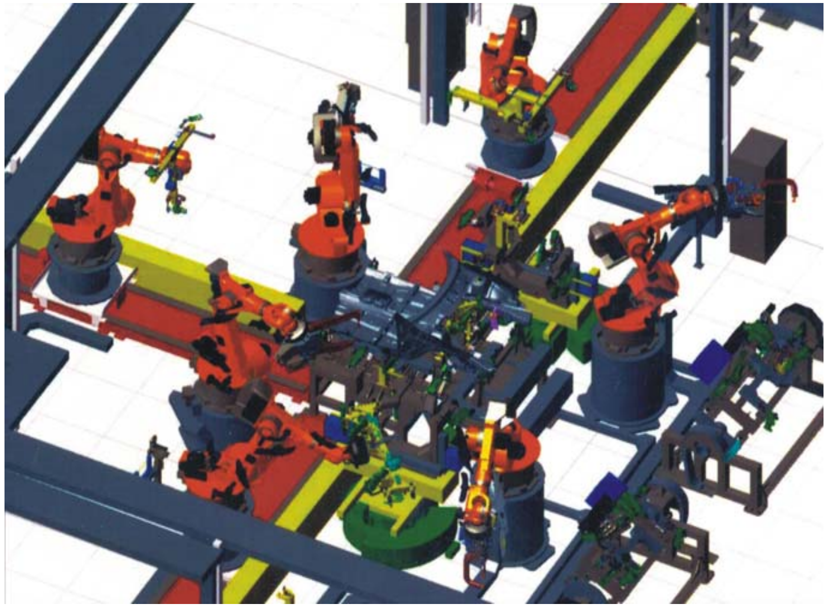
$$\Sigma_u(\delta_P(i_P, s)) \subseteq \Sigma_u(\delta_{P\|K}(i_{P\|K}, s))$$

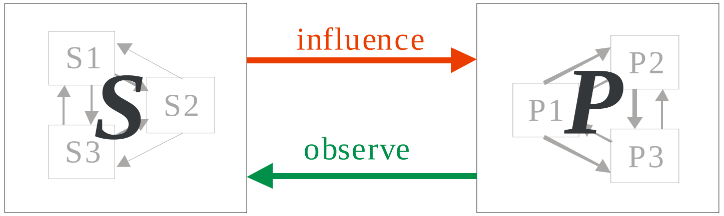
Supervisor - Synthesis



- Algorithm
  1. Calculate T_0 = P||K
  2. Find un-controllable states $S_0 = f(P, T_0)$
  3. $S_{i+1} = SupNB(S_i)$
  4. $S_{i+2} = SupC(S_{i+1})$
  5. If $S_{i+2} \neq S_{i+1}$, go to 3
  6. $S := S_{i+1}$
- Claim:
  - Within spec
  - Non-blocking
  - Controllable
  - **Maximally permissive**

Supervisor – Monolithic Synthesis





- Process typically described by
  - Interacting **sub-processes**
  - $P = P_1||P_2||\ldots||P_n$
  - Restrict each other
- Spec typically described by
  - Interacting **sub-specs**
  - $K = K_1||K_2||\ldots||K_m$
  - Restrict each other
- **Monolithic** supervisor
  - Single one for the **entire** *P* and **entire** *K*
- Guarantees
  - **No** specs violated
  - But...