Sequential Processes

Sequential Process = Sequence of *actions*

- Finite set of actions
- Possibly infinite sequence

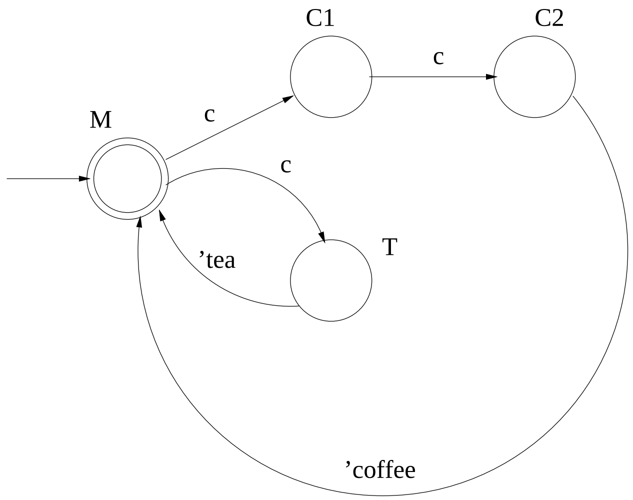
Finite description?

Automata

- a set of states
- the start state
- the accepting states
- the transitions

The language = the accepted sequences of transitions

Automata



c 'tea

c c 'coffee

c 'tea c 'tea

c 'tea c c 'coffee

...

Regular Sets

Operations on languages:

- union $S_1 \cup S_2$ (denoted $+$)
- concatenation $S_1 S_2 = \{s_1 s_2 \mid s_1 \in S_1, s_2 \in S_2\}$
- star (iteration) $S^* = \epsilon \cup S \cup SS \cup \ldots$

S over A is regular iff

- is empty, or
- is a singleton $\{a\}$ where $a \in A$, or
- can be obtained from a regular set by union, concatenation or iteration.

Regular Sets: Algebraic Laws

$$(S_1\ S_2)\ S_3 \ = \ S_1\ (S_2\ S_3)$$
$$(S_1 + S_2)\ T \ = \ S_1\ T \ + \ S_2\ T$$
$$T\ (S_1 + S_2) \ = \ T\ S_1 \ + \ T\ S_2$$
$$S\ \epsilon \ = \ S$$
$$S\ \emptyset \ = \ \emptyset$$
$$S\ (T\ S)^* \ = \ (S\ T)^*\ S$$

Arden's Lemma

For any sets of strings S and T the equation

$$X = S\ X + T$$

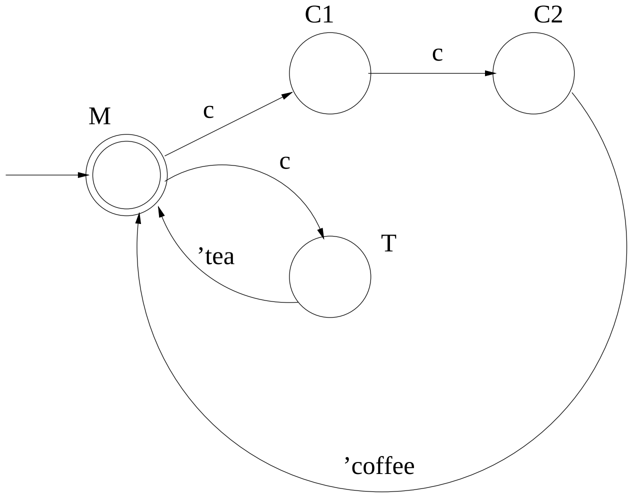
has a solution

$$X = S^* T$$

The solution is unique if $\epsilon \notin S$

Arden's Lemma

Finite state systems



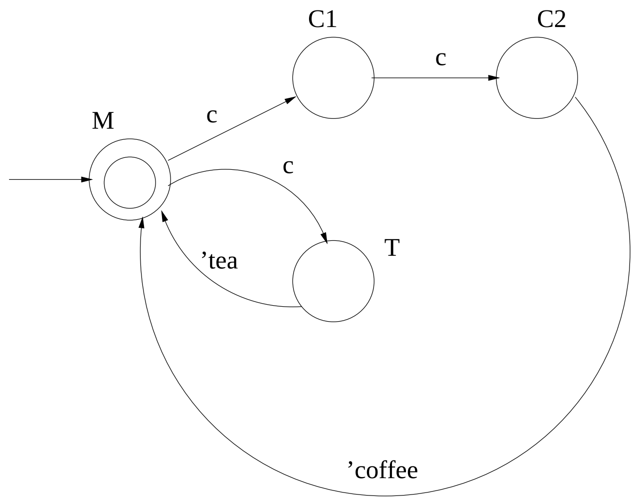
M = c C1 + c T + ε
T = 'tea M
C1= c C2
C2 = 'coffee M

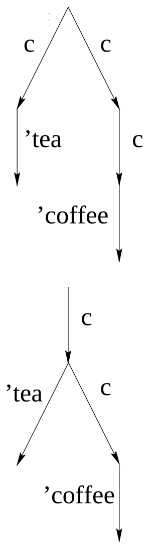
M = c c 'coffee M + c 'tea M + ε

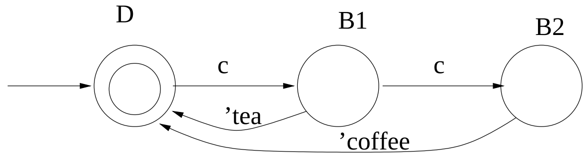
M = c (c 'coffee + 'tea) M + ε

M = (c (c 'coffee + 'tea))* ε = (c (c 'coffee + 'tea))*

DFA vs. NFA

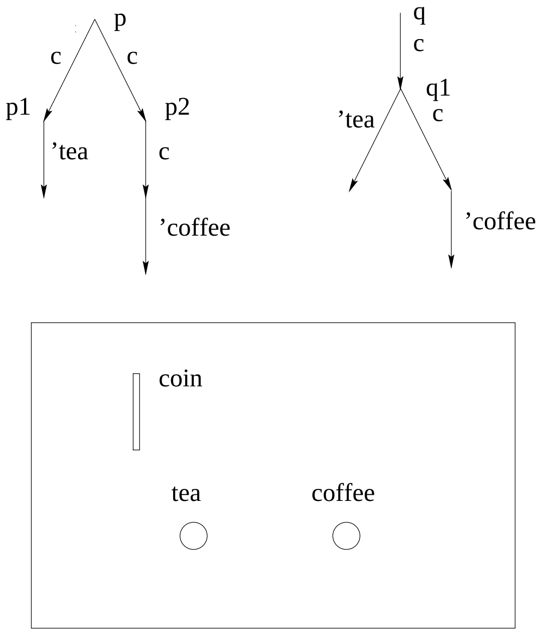






Reactive Systems

The environment controls the actions: some are possible, some not



Labelled Transition System

- a set of *states* $\mathcal{Q}$
- a set of *actions* Act : *names* and *co-names*
- *transition relation* $\subseteq \mathcal{Q} \times Act \times \mathcal{Q}$
  Notation: $p \rightarrow_a q$

LTS = Finite Automaton without start state or accepting states

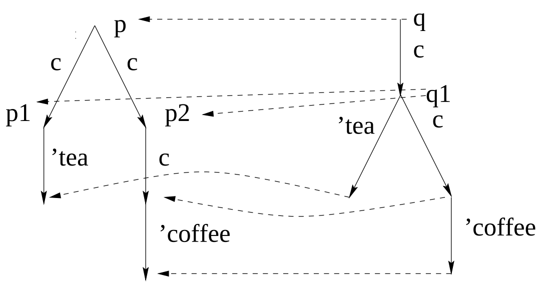
When states p, q are equivalent?

- allow the same actions
- on each action allowed it should be possible to reach equivalent states

Strong Simulation

Given LTS $M = (Q, T)$. A **strong simulation** over M is a binary relation S on Q s. that
whenever pSq:

$$p \to_a p' \;\Rightarrow \exists q' : q \to_a q' \; and \; p'Sq'$$

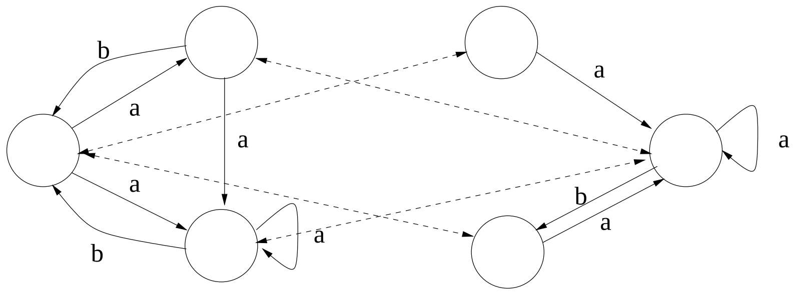


qSp

Strong Bisimulation

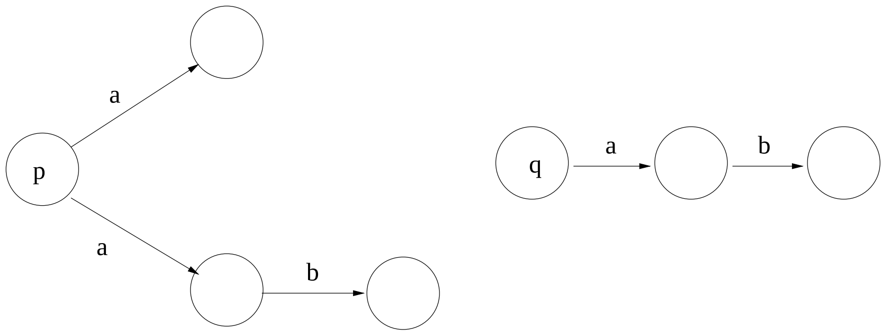
S is a *strong bisimulation* iff S and S^{-1} are strong simulations.

$p \sim q$ iff there exists a strong bisimulation S s.that pSq
$\sim$ is an **equivalence** relation



Strong Bisimulation

Are there strong simulations betw. p and q?
Are p and q bisimilar?



Sequential Process Expressions

- empty process `0`
- Actions/ action parameters `a,b,...`
- (Parameterized) process names `A(a,b)`

- Sequential process expression:

$$P ::= A(...) \mid \sum_i \alpha_i.P_i$$

  process name with parameters, or
  sum of i prefix expressions
  $i \geq 0$

- Definition for every process name
  process name = sum of prefix expr.
  (all action names appear as parameters)

```
P0(x,y) = x.P1(x,y) + x.P2(x,y)
P1(u,v) = u.P2(u,v) + v.P0(u,v)
P2(r,s) = r.P2(r,s) + s.P0(r,s)
```

Checking Bi-similarity of Sequential Processes

```
proc C = a.C1
proc C1 = b.C

proc D = a.b.D
```

```
cwb-nc> eq -S bisim C D
Building automaton...
States: 4
Transitions: 4
Done building automaton.
TRUE
```


Structural Congruence on Process Expressions

$$P \equiv Q$$

based on process definitions.

```
A(a,b) = a.A(a.b) + b.B(a,a)
B(c,d) = c.d.0
```

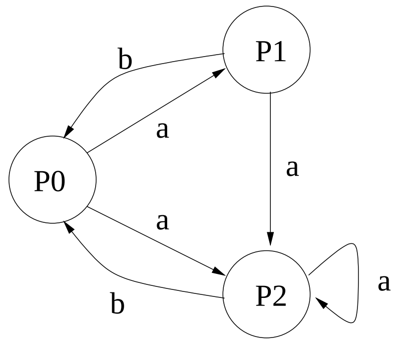
B(a,a) $\equiv$ a.a.0
A(a,b) $\equiv$ a.(a.A(a,b)+b.B(a,a)) + b.a.a.0

P $\equiv$ Q iff P,Q can be transformed to each other by "using" definitions:
replace `A(b)` by {`b/a`}`P`, or v.v.
where `A(a) = P` is a definition.

LTS's of sequential processes

States = Process Expressions
`P` $\to_a$ `Q` iff `P = a.Q + ...`

P0(x,y) = x.P1(x,y) + x.P2(x,y) [a,b/x,y]
P1(u,v) = u.P2(u,v) + v.P0(u,v) [a,b/u,v]
P2(r,s) = r.P2(r,s) + s.P0(r,s) [a,b/r,s]



Boolean buffer of capacity n

- Actions: `in0, in1, 'out0, 'out1`
- States: `B, B0, B1, B00, B01, B10, B11`
- Transitions:

```
B   = in0.B0 + in1.B1
B0  = 'out0.B + in0.B00 + in1.B10
B1  = 'out1.B + in0.B01 + in1.B11
B00 = 'out0.B0
B01 = 'out1.B0
B10 = 'out0.B1
B11 = 'out1.B1
```

A Scheduler for n tasks

- Task i
  begins with `ai`
  ends with `bi`
- `a1,...,an` occur cyclically
- `bi` must precede next `ai`
- No other restrictions

Describe this as a sequential process with actions `ai, bi`.

A Scheduler for 2 tasks

```
Sc    = Sc100

Sc100 = a1.Sc210
Sc101 = a1.Sc211 + b2.Sc100
Sc110 = b1.Sc100
Sc111 = b1.Sc101 + b2.Sc110

Sc200 = a2.Sc101
Sc201 = b2.Sc200
Sc210 = a2.Sc111 + b1.Sc200
Sc211 = b1.Sc201 + b2.Sc210
```