An algorithm for Bisimilarity on finite-state LTSs

- Establishing bisimilarity is a special case of a more general problem, called *Relational Coarsest Partition Problem* (RCPP).
- RCPP is based on refining a set into classes (or blocks) by starting from the coarsest possible partition (i.e., the set itself), by separating elements that are not equivalent (according to a specific rule)
  - Start with the corset possible partition (i.e., the given set);
  - Split the blocks of the partition (by using the splitting rule) until possible;
  - OUTPUT: two states are in relation if and only if at the end they belong to the same block.
- The key feature is the rule that decides the splitting of blocks in sub-blocks

  → This characterizes the different equivalences that can be obtained by using the algorithm for RCPP.

- Let B be a generic block of a partition and s and t be two states belonging to B
- Let us call block(s) the set of states belonging to the same block as s.
- According to bisimilarity, s and t must be separated (and, hence, B must be split) iff:

$$\exists\, \alpha, s' \text{ s.t. } s \xrightarrow{\alpha} s' \quad \wedge \quad \forall t' (t \stackrel{\alpha}{\nrightarrow} t' \vee t' \notin block(s'))$$

- We shall use an alternative condition (equivalent to the splitting rule given above).
- Define $tgt_\alpha(s) = \{B' : \exists\, s'.\ s \text{ -}\alpha\text{-> } s' \wedge s' \in B'\}$
- The alternative splitting condition is:

$$\exists\, \alpha\ .\ tgt_\alpha(s) \neq tgt_\alpha(t)$$

Main Algorithm

INPUT: $LTS=(S,\Lambda,\rightarrow)$, $s,t \in S$

```
        P←{S}
LOOP:   forall B∈P do
            for all α ∈ Λ do
                I ← split(B, α, P)
                if I ≠ {B} then
                    P ← (P \ {B}) ∪ I
                    goto LOOP
        forall B∈P do
            if {s,t} ⊆ B then return true
        return false
```

Splitting Algorithm

INPUT: (B,α,P)

$s \in B$

$B1 \leftarrow \varnothing \quad B2 \leftarrow \varnothing$

forall $t \in B$ do

if $tgt_\alpha(s) = tgt_\alpha(t)$ then $B1 \leftarrow B1 \cup \{t\}$

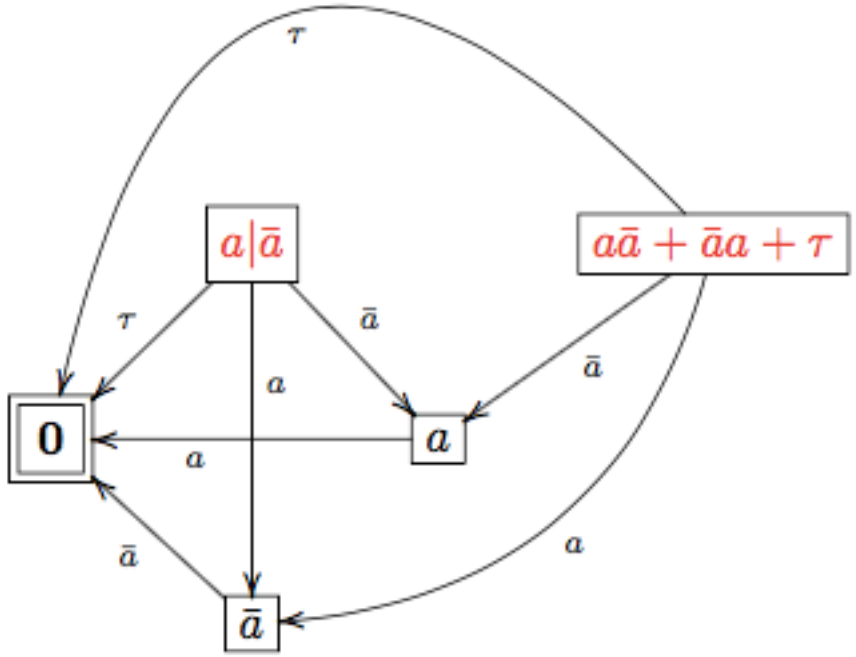
else $B2 \leftarrow B2 \cup \{t\}$

if $B2 = \varnothing$ then return $\{B\}$

else return $\{B1, B2\}$

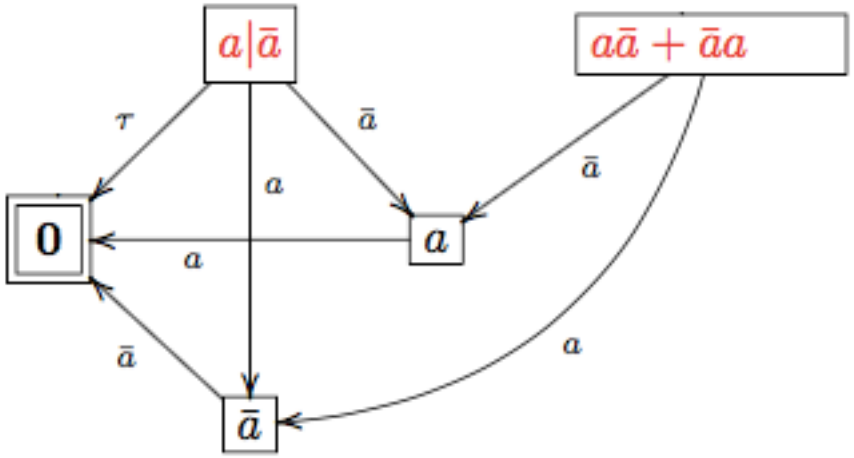
An Example

Assigned Value	State
1	$a\|\bar{a}$
2	$a\bar{a} + \bar{a}a + \tau$
3	$\bar{a}$
4	a
5	$\mathbf{0}$



- P = {{1,2,3,4,5}}:
  - Choose a and B = {1,2,3,4,5}
  - $tgt_a(1) = tgt_a(2) = tgt_a(4) = \{B\}$; $tgt_a(3) = tgt_a(5) = \varnothing$
  - split(B, a, P) replaces B by two sub-blocks in P
- P = {{1,2,4},{3,5}}:
  - Choose $\underline{a}$ and B = {1,2,4}
  - $tgt_{\underline{a}}(1) = tgt_{\underline{a}}(2) = \{B\}$, whereas $tgt_{\underline{a}}(4) = \varnothing$
  - split(B,$\underline{a}$,P) replaces B by two sub-blocks in P
- P = {{1,2},{4},{3,5}}:
  - Choose $\underline{a}$ and B = {3,5}
  - $tgt_{\underline{a}}(3) = \{B\}$ whereas $tgt_{\underline{a}}(5) = \varnothing$
  - split(B,$\underline{a}$,P) replaces B by two sub-blocks in P
- With P = {{1, 2}, {4}, {3}, {5}}, every block and action lead to no other split
- Because 1 and 2 are in the same block, $a|\underline{a}$ and $a\underline{a}+\underline{a}a+\tau$ are bisimilar

- The algorithm can also be used to prove that two processes are NOT equivalent, e.g.



- Like before, we arrive at P = {{1, 2}, {4}, {3}, {5}}
- But now, if we choose B = {1, 2} and action τ, we can further split B into {1} and {2}
- So, a|a and aa+aa are NOT bisimilar

Correctness Proof

- The algorithm always terminates
  - every cycle works on a finite number of elements
  - the cardinality of P is at most |S| (one state in every block)
  - in this case, no block can be further split

    → this ensures termination!
- Let P_f be the partition at the end of the algorithm; then,

$$s \sim t \text{ IFF } \exists B \in P_f \,.\, \{s,t\} \subseteq B$$

IF:

- Hyp: $s \sim t$ Th: $\exists B \in P_f \,.\, \{s,t\} \subseteq B$
- By contradiction, s and t are in different blocks of the final partition
- At the beginning of the algorithm, P had a single block containing all the states
- s and t have been divided in a certain step of the algorithm, say the (i + 1)-th
  - → at step i, there was a partition $P_i = \{B_1, B_2, \ldots, B_k\}$ such that $\exists\ j : 1 \le j \le k \,.\, \{s,t\} \subseteq B_j$
  - → $P_{i+1} = \{B_1, B_2, \ldots, B_j', B_j'', \ldots, B_k\}$ with $s \in B_j'$ and $t \in B_j''$
- If s and t have been separated, $\exists\, \alpha \,.\, tgt_\alpha(s) \neq tgt_\alpha(t)$
  - →

$$\exists\, s' : s \xrightarrow{\alpha} s' \ \wedge\ \forall\, t' (t \overset{\alpha}{\not\to} t' \vee block(s') \neq block(t'))$$

$$\exists\, t' : t \xrightarrow{\alpha} t' \ \wedge\ \forall\, s' (s \overset{\alpha}{\not\to} s' \vee block(t') \neq block(s'))$$

- This contradicts $s \sim t$

ONLY IF:

- Hyp: $\exists B \in P_f$. $\{s,t\} \subseteq B$ Th: s~t
- Define $R = \{(x, y) \in B \times B : \exists B \in P_f\}$
- we prove that R is a bisimulation (notice that {s,t} belongs to R by construction)
- It is symmetric, so it is enough that is a simulation
- Let $(x,y) \in R$ and $x \xrightarrow{\alpha} x'$
- The block of x is no more splittable (the algorithm has terminated)
  - → $tgt_\alpha(x) = tgt_\alpha(y)$
  - → $\exists y' : y \xrightarrow{\alpha} y'$ and block(x′) = block(y′)
  - → x′ and y′ belong to the same block in P_f
  - → By construction, $(x', y') \in R$

Complexity

- Let (S,Λ,−>) be an LTS with |S|=n and |−>|=m
- The final loop costs O(n)
- The loop *LOOP* is executed every time there is a partitioning

  → at most O(n) times
- For the nested for-loops:
  - One loop for every block B∈P
  - One loop for every label α∈Λ
  - The check I ≠ B can be performed in constant time, so as operation

    P ← (P \ {B}) ∪ I
  - So the real cost is due to function *split*

```
      P←{S}
LOOP: forall B∈P do
        for all α ∈ Λ do
          I ← split(B, α, P)
          if I ≠ {B} then
            P ← (P \ {B}) ∪ I
            goto LOOP
      forall B∈P do
        if {s,t} ⊆ B then return true
      return false
```

- One loop for every state t belonging to the chosen block B
- The check $tgt_\alpha(s) = tgt_\alpha(t)$ performed for the chosen states s and t
  - → We shall prove that this check can be performed by analyzing every α-trans. from states belonging to B only once
- So, the overall cost of the nested for-loops is

```
s∈B
B1 ← ∅   B2 ← ∅
forall  t∈B  do
    if tgtα(s) = tgtα(t) then B1 ← B1 ∪ {t}
                         else B2 ← B2 ∪ {t}
if B2 = ∅ then return {B}
          else return {B1, B2}
```

$$\sum_{B\in\mathcal{P}}\sum_{\alpha\in\Lambda}\sum_{t\in B}|\{t' : t \xrightarrow{\alpha} t'\}| = \sum_{B\in\mathcal{P}}\sum_{t\in B}\sum_{\alpha\in\Lambda}|\{t' : t \xrightarrow{\alpha} t'\}| = \sum_{t\in S}\sum_{\alpha\in\Lambda}|\{t' : t \xrightarrow{\alpha} t'\}| = |\rightarrow| = m$$

- Thus, the total cost is O(nm)

On efficiently handling blocks

- Symbolic representation of blocks
- Every block is a list of states and is associated to a binary string (+ lexicographic ordering)
- The starting block contains all states and has index 0
- When a block is split, the resulting sub-blocks are associated to the string of the original block with a new final bit, respectively 0 and 1

  Example:
  - let S = {1, 2, 3, 4, 5} have index 0
  - the first split produces (B1 = {1, 2, 4}, B2 = {3, 5})
  - Their associated binary strings will be 00 (for B1) and 01 (for B2)
  - If then B2 is split into (B3 = {3}, B4 = {5}), then the associated strings will be 010 (for B3) and 011 (for B4)

- Then, we can consider the transition relation as a collection of triples (s,α,σ), whenever $s \xrightarrow{\alpha} s'$, $s' \in B$ and σ is the binary index of B
- We fix an ordering on states and actions and lexicographically order the triples above
  - At the beginning this costs O(mlogm)
  - Every update of a block entails updating the triples that lead to that block (REMARK: this does NOT require a new ordering!)
- We obtain a matrix (m×3) that looks like this:

$(s_1 \quad \alpha_{1,1} \quad \sigma_{1,1})$
$(s_1 \quad \alpha_{1,2} \quad \sigma_{1,2})$
…
$(s_1 \quad \alpha_{1,k1} \quad \sigma_{1,k1})$
$(s_2 \quad \alpha_{2,1} \quad \sigma_{2,1})$
…
$(s_2 \quad \alpha_{2,k2} \quad \sigma_{2,k2})$
…
$(s_n \quad \alpha_{n,1} \quad \sigma_{n,1})$
…
$(s_n \quad \alpha_{n,kn} \quad \sigma_{n,kn})$

Example:

…
$(s_1 \quad \alpha_1 \quad 00)$
$(s_1 \quad \alpha_2 \quad 01)$
$(s_1 \quad \alpha_2 \quad 11)$
…

If we split 01 (into 010 and 011), wherever s1 has to go, the matrix will be

…
$(s_1 \quad \alpha_1 \quad 00)$
$(s_1 \quad \alpha_2 \quad 01b)$
$(s_1 \quad \alpha_2 \quad 11)$

If in Split(B,α,P) we fix s as pivot state, then the check $tgt_\alpha(s) = tgt_\alpha(t)$ (for all t in B) is done as follows:

- Let k be the number of α-transitions from s
- If k=0, then

  B1 ← {x∈B : x cannot perform α}

  B2 ← {x∈B : x can perform α}
- Otherwise, let σ be the index of the first α-trans for s; For every t∈B

  if the index of the first α-trans for t is σ

  then B1 ← B1∪{t} else B2 ← B2∪{t}
- Let σ' be the index of the second α-trans for s; For every t∈B1

  if t has not a second α-trans or the index of its second α-trans is not σ'

  then B1 ← B1\{t} and B2 ← B2∪{t}
- …
- Let σ'' be the index of the kth α-trans for s; For every t∈B1

  if t has not a kth α-trans or the index of its kth α-trans is not σ''

  then B1 ← B1\{t} and B2 ← B2∪{t}
- For every t∈B1

  if t has other α-trans

  then B1 ← B1\{t} and B2 ← B2∪{t}