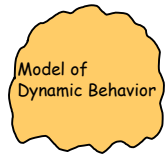
Lecture 1

Infinite-state and parameterized systems

Model Checking



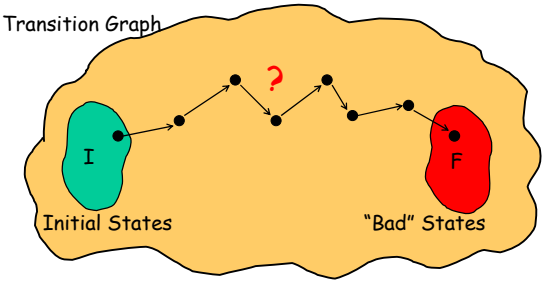
?
Satisfies

Correctness Property

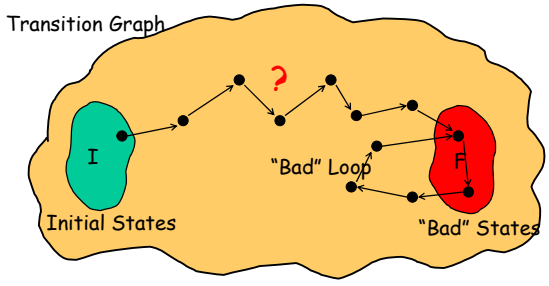
- Distributed Algorithm
- Communication Protocol
- Model of Software Module
- Design Model

- Assertions
- Temporal Logic Formula
- Abstract Test

Reachability Problem:



Repeated Reachability Problem:



Verification of Finite-State Systems

- A rather mature technology
- Efficient representations
  - Hash tables
  - Symbolic representations: BDDs, SAT-based
- Exploitations of Model structure
  - Concurrency (partial order reduction)
- Partial approximations
  - Symbolic Trajectory Evaluation
  - Exploration of part of State-Space.

Infinite-State Systems

Why are Models infinite-state?

- Unbounded data structures
  - Stacks, Queues, Numbers, …
- Unbounded Control Structures
  - Process Networks, Distributed Algorithms, Dynamic Process Creation
- Clocks
  - Timed and hybrid Systems
- Unbounded Communication Channels
- …

Some classes of Infinite-State Models

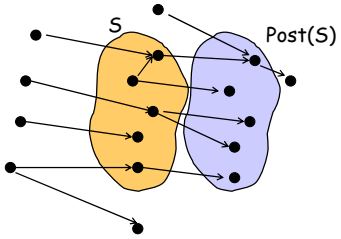
Extending State Vectors by

- Integer Variables
  - Petri Nets, Programs with Integers
  - Unstructured Process Networks.
- Strings over Finite Alphabet
  - Linear Data Structures (Stacks, Queues, Lists)
  - Numbers
  - Linear/Ring-shaped process networks
- (labeled) Graphs
  - General Distributed Algorithms,
  - Heaps, …

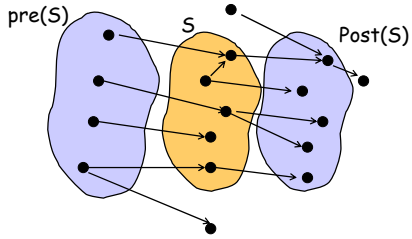
Recepy for Infinite-State Verification

- Choose symbolic representation of (infinite) sets of system states.
- Sets of sucessors/predecessors computed by applying predicate transformers
- Explore state-space by repeatedly computing sets of successors/predecessors of already explored states.
- Safety properties reduces to reachability
- Liveness reduces to repeated reachability (at least in some cases).

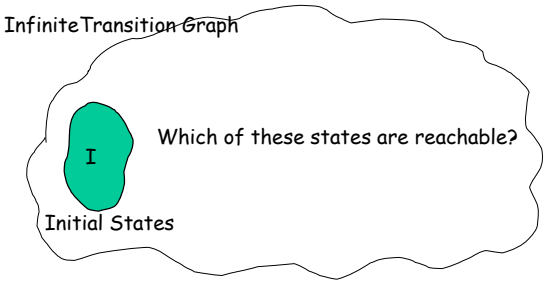
Pre- and PostConditions



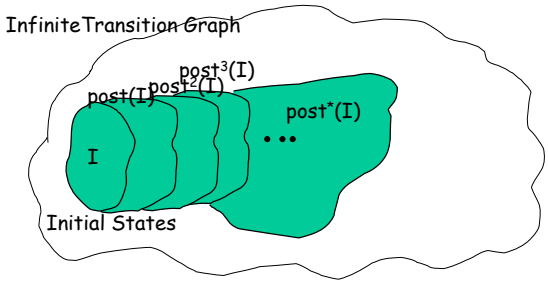
Pre- and PostConditions



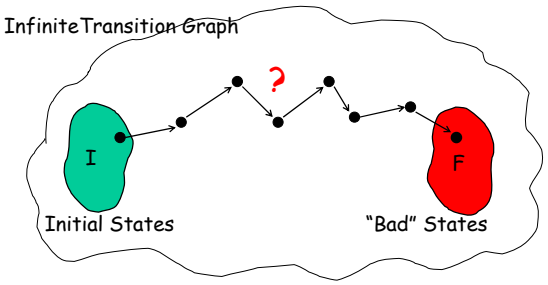
Generating strong(est) invariant:



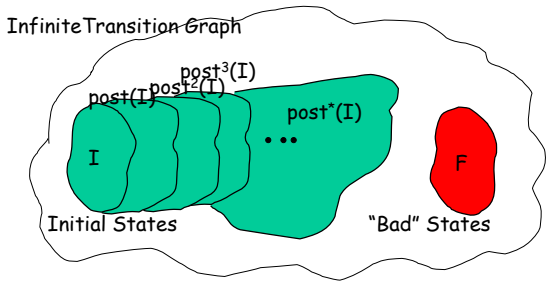
Generating strong(est) invariant:



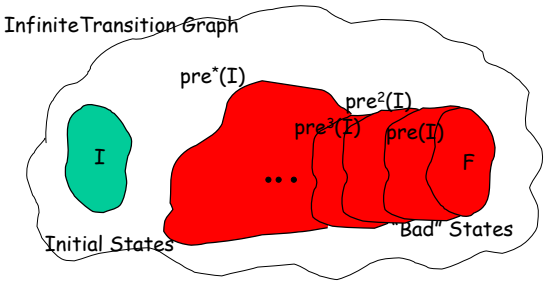
Reachability Problem:



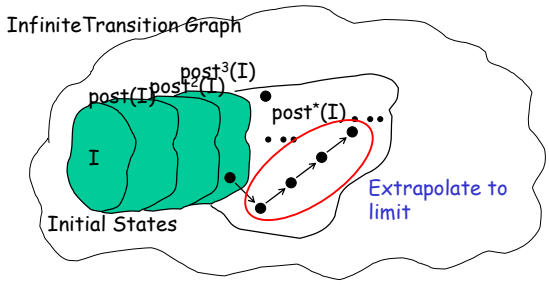
Forward Reachability Analysis:



Backward Reachability Analysis:

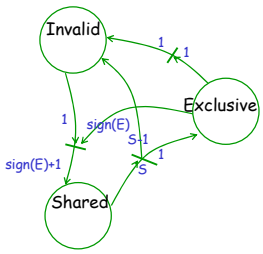


Acceleration/Widening



Process Networks: Simple Cache Protocol

Combine all Processors for one Cache line.



Initially: S = E = 0
Actions
I ≥ 1 → I := I-1
S := S + sign(E) + 1
E := E - sign(E)
S ≥ 1 → I := I + S -1
S := 0
E := E + 1
E ≥ 1 → I := I+1
E := E-1

Backwards Reachability Analysis

Bad: E ≥ 2
Pre(r-m,Bad): E ≥ 1, S ≥ 1 φ
Pre(r-s,Bad): E ≥ 3
Pre(inv,Bad): E ≥ 3
Pre(r-m, φ): false
Pre(r-s, φ): E ≥ 2 , I ≥ 1
Pre(inv, φ): E ≥ 2 , S ≥ 1

Pre*(Bad): E ≥ 2 ∨ E ≥ 1, S ≥ 1
Unreachable!!

Initially: S = E = 0
Actions
I ≥ 1 → I := I-1
S := S + sign(E) + 1
E := E - sign(E)
S ≥ 1 → I := I + S -1
S := 0
E := E + 1
E ≥ 1 → I := I+1
E := E-1

The fact that backwards reachability analysis converged is not an accident.

It is because the model is a well-structured transition system

[Finkel][Abdulla,Cerans,Jonsson,Tsay]

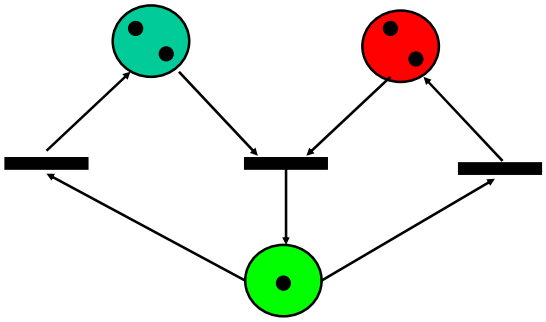
Well-Structured Transition Systems

- There is partial ordering $\preceq$ on configurations

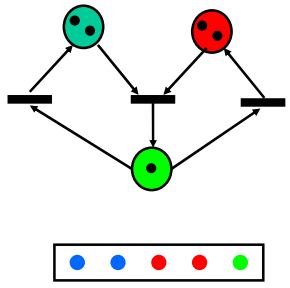
  $\{I \to 0, S \to 1, E \to 2\} \preceq \{I \to 2, S \to 1, E \to 3\}$

  such that
  - A: Transition relation is monotone wrp. to $\preceq$
  - B: $\preceq$ on configurations is a well-quasi-ordering: Any UCS has finite set of minimal elements
- The set Bad is Upward Closed (UCS)
- by A: Bad is UCS implies pre(Bad) is UCS implies pre*(Bad) is UCS

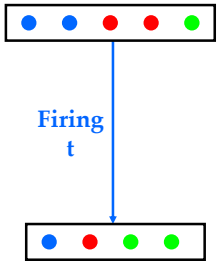
Petri Nets

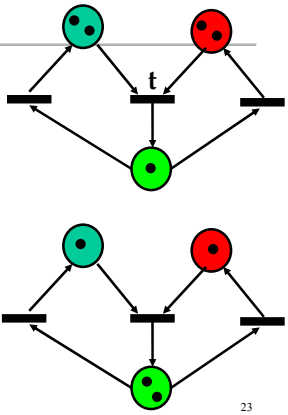


States = Markings



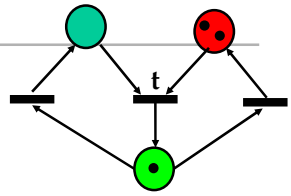
Transitions



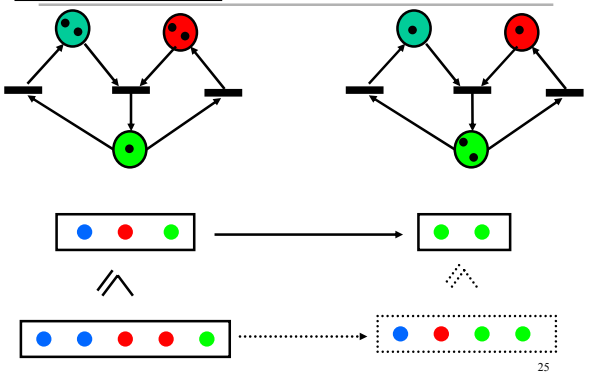


Transitions

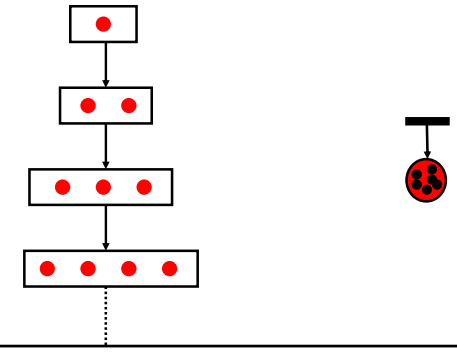
t is disabled



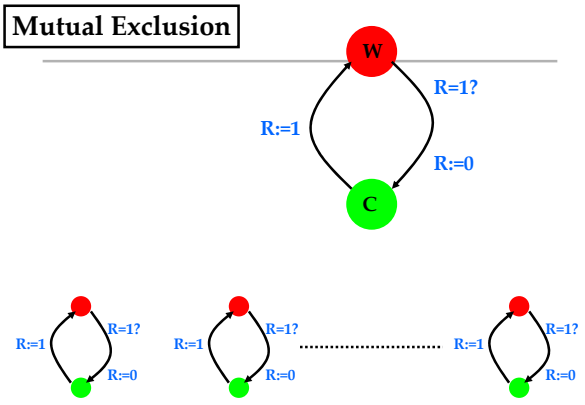
Monotonicity



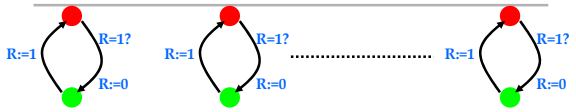
Petri Nets: infinite state



Mutual Exclusion



Mutual Exclusion

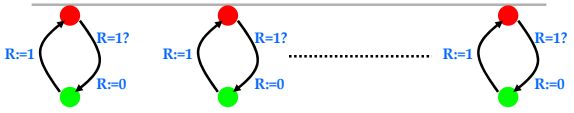


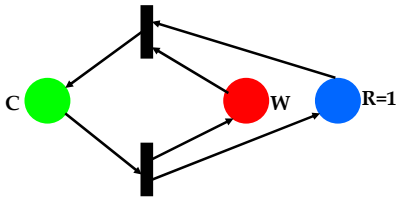
Initial states:
- R=1
- All processes in

Infinitely many

Bad states:
Two or more processes in

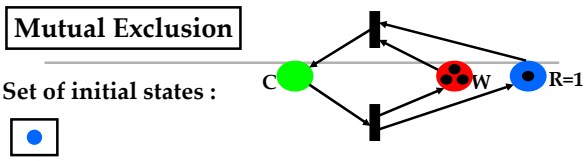
Mutual Exclusion

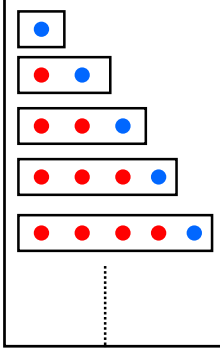




Mutual Exclusion

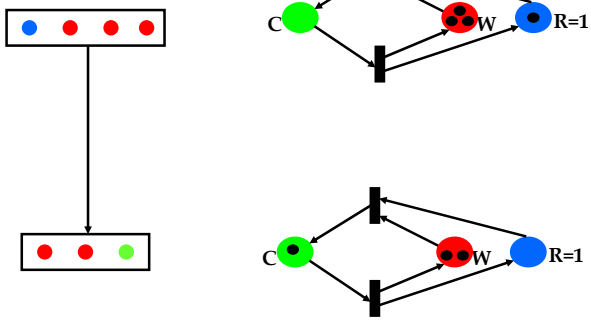
Set of initial states :



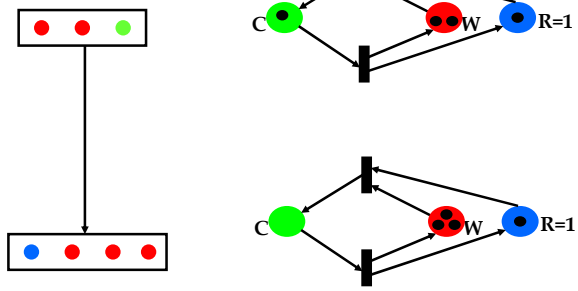


infinite

Mutual Exclusion



Mutual Exclusion

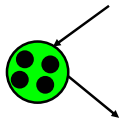


Safety Properties

- mutual exclusion:
  #tokens in critical section > 1

Ideal = Upward closed set of markings

critical section



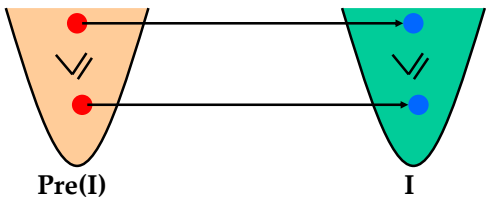
safety
=
reachability of ideals

Petri Nets

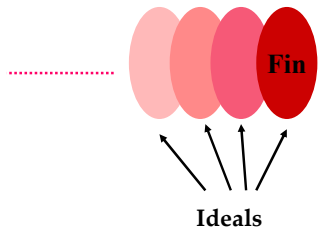
- Concurrent systems
- Infinite-state: symbolic representation
- Monotonic behaviour
- Safety properties: reachability of ideals

Monotonicity

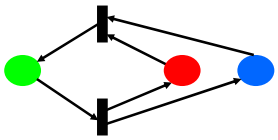
ideals closed under computing Pre

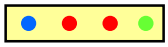


Backward Reachability Analysis

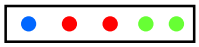


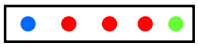
Ideals: Symbolic Representation

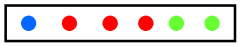




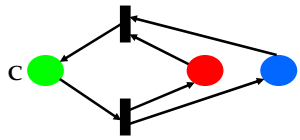
index (generator)







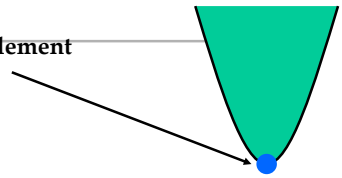
Ideals: Symbolic Representation



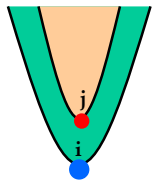
Index for bad states

Each ideal can be characterized by a finite set of generators

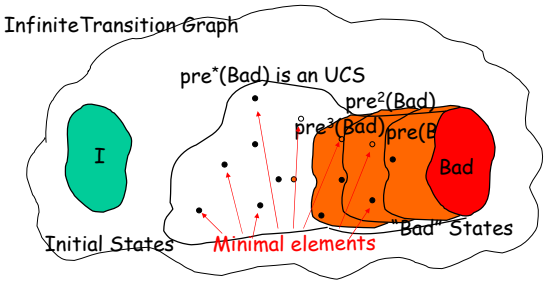
Index is minimal element of its ideal



If $i \leqslant j$ then

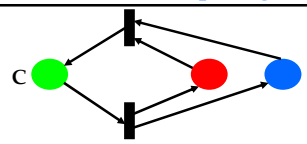


Backward Exploration [Abdulla Jonsson 93]



•Eventually, all minimal elements reached => termination

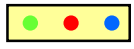
Monotonicity → ideals closed under computing Pre

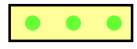


Index for bad states

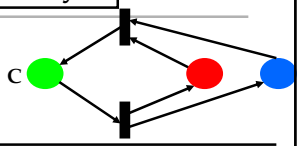
i: index
Pre(i) computable

Indices of Pre





Backward Reachability Analysis

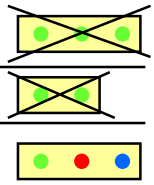


Step 0 :

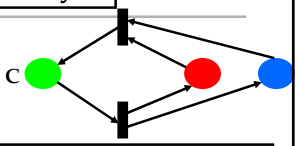
Step 1 :

Step 2 :

Step 3 :



Backward Reachability Analysis

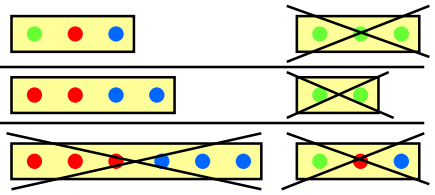


Step 0 :

Step 1 :

Step 2 :

Step 3 :



What did we need?

1. Computable ordering
2. Monotonicity, Computability of Pre
3. Termination -- Ordering is WQO

↑

"nice properties"

Well Quasi-Ordering (WQO)

$(A, \sqsubseteq)$ is WQO if

$\forall a_0\ a_1\ a_2\ a_3$ .....

$\exists i,j:\ i<j$ and $a_i \sqsubseteq a_j$

WQO : Simple Example

$(Nat, \leq)$ is WQO

$\forall x_0\ x_1\ x_2\ x_3$ ..... : natural numbers

$\exists i,j:\ i<j$ and $x_i \leq x_j$

Properties of WQO

Finite Sets

$(A, =)$ is WQO if A is finite

$a_0\ a_1\ a_2\ b\ a_3\ a_4\ a_5\ b\ a_6$ .....

Examples -- WQO

$(A^*, \sqsubseteq)$

A : finite alphabet

$w_1 \sqsubseteq w_2$: w_1 subword of w_2

e.g. ab $\sqsubseteq$ xaybz

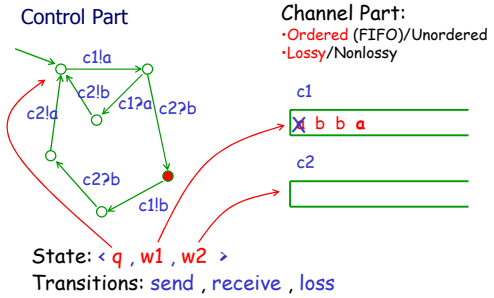
Why are words over a finite set WQO?

[Higmans lemma]
Proof by contradiction:

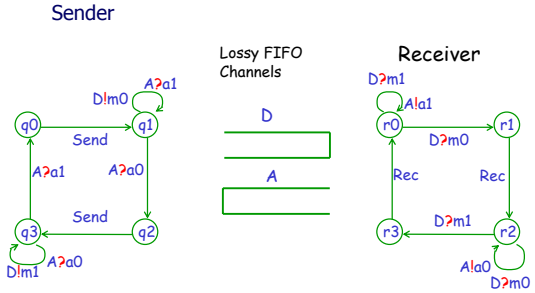
- Assume that there is a sequence of words
  w0 w1 w2 w3 w4 w5 w6 w7 . . .
  where never $w_i \leq w_j$ for $i < j$
- Let this be the least such sequence lexicographically, meaning that
  - w0 is as short as possible
  - if w0 w1 w2 w3 are fixed, then w4 is as short as possible
- There must be infinite subsequence starting with, say, a
  w0 w1 aw2' w3 aw4' w5 w6 aw7' . . .
- But then
  w0 w1 w2' w4' w7' . . .
  is a smaller such sequence

Contradiction

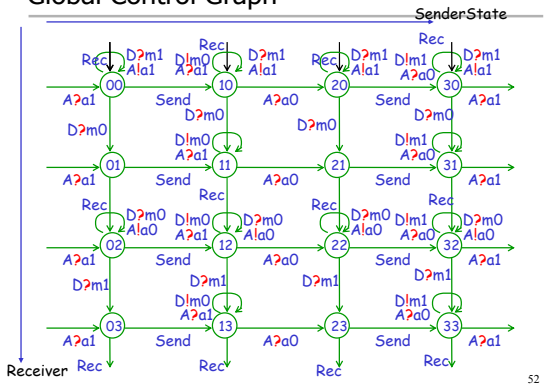
Protocol Model



Alternating Bit Protocol



Global Control Graph



Monotonicity

$< q_i, r_j, w_D, w_A >$ ⟶ $< q_i, r_j, w_D, w_A >$

$\sqsubseteq$

$< q_i, r_j, w_D', w_A' >$

Since messages can be lost.

Ideal Index

$< q_i, r_j, w_D, w_A >$ **denotes all larger states**

< q0 , r0 , <m0> , <> > ⟶

< q0 , r0 , <m0 m1> , <> >

< q0 , r0 , <m0 m1> , <a0 a0 a1> >

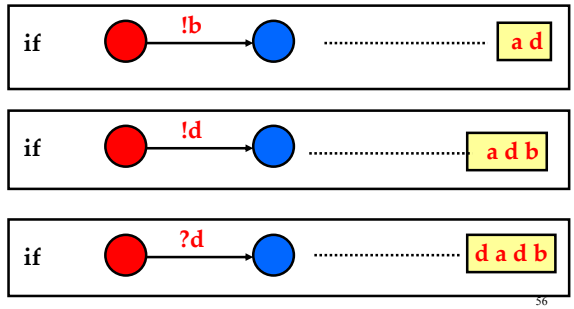
< q0 , r0 , <> , <a1 a0 a1> >

Each ideal can be characterized by a finite set of generators

By WQO of $\sqsubseteq$

Example

Pre (a d b)



Methodology (applied to LCS)

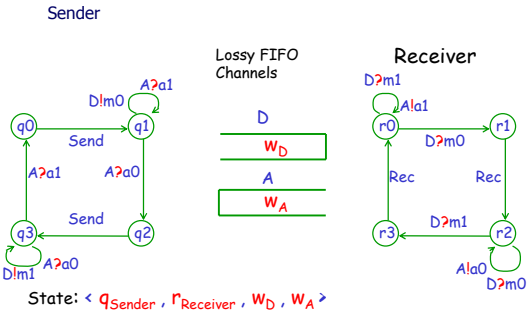
1. Computable ordering
2. Monotonicity, Computability of Pre
3. Ordering is WQO

LCS -- Forward vs Backward Analysis

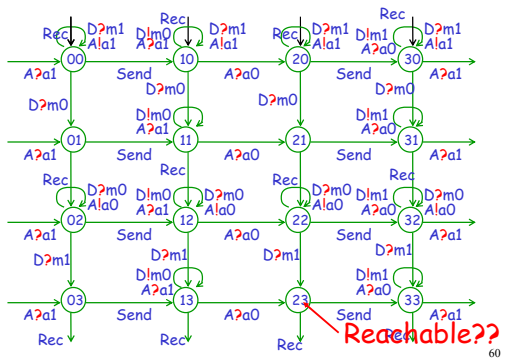
Pre*(w) is regular and computable

Post*(w) is regular but not computable

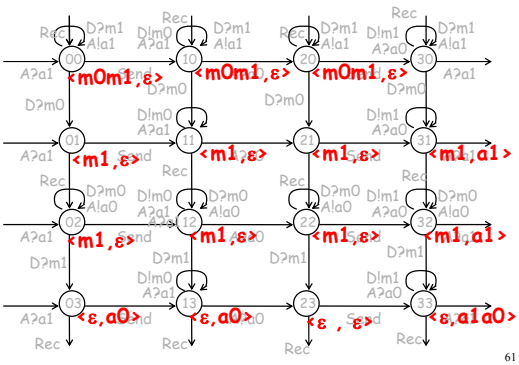
Alternating Bit Protocol



Analysis of AB Protocol



Analysis of AB Protocol



Methodology

- Start from a finite domain
- Build more complicated data structures: words, multisets, lists, sets, etc.

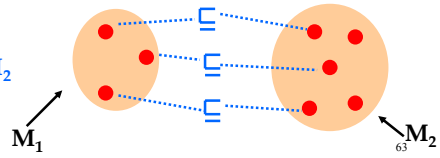
Properties of WQO

Multisets

if $(A, \sqsubseteq)$ is WQO

then $(A^M, \sqsubseteq^M)$ is WQO

$M_1 \sqsubseteq^M M_2$



Properties of WQO

Words

if $(A, \sqsubseteq)$ is WQO

w_1 : a_0 a_1 a_2

$\sqcap_*$ $\sqcap$ $\sqcap$ $\sqcap$

w_2 : b_0 b_1 b_2 b_3 b_4 b_5 b_6

then $(A^*, \sqsubseteq^*)$ is WQO

Examples -- WQO

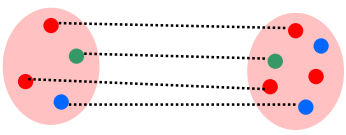
Words of natural numbers

5 2 7 ← w_1

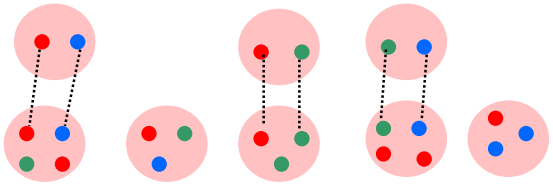
3 7 1 4 2 8 ← w_2

$w_1 \sqsubseteq w_2$

Multisets over a finite alphabet



Words of multisets over a finite alphabet

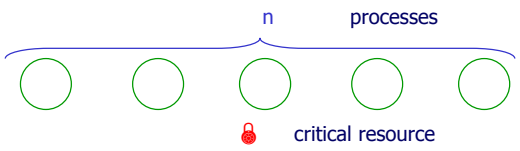


Parameterized Systems

Family of Systems
- infinite number of parameter values
- Each system instance can be finite-state
- Examples: Process Networks, Distributed Algorithms
  - Parameter: system size, system topology, …
  - Example: Linear system parameterized by its size

Parameterized Systems

Family of Systems
- infinite number of parameter values
- Each system instance can be finite-state
- Examples: Process Networks, Distributed Algorithms
  - Parameter: system size, system topology, …
  - Example: Linear system parameterized by its size



Verifying Parameterized Systems

Undecidable in general [Apt, Kozen]

Different Approaches:
- Verify system for some parameter values
  - Sometimes, results can be generalized
- Induction over system structure [Kurshan, Wolper/Lovinfosse]
  - Must find Inductive Hypothesis, "Network Invariant".
- Generate Finite-State Abstraction
- Extend Symbolic Model Checking to Infinite Sets
  - Try to compute reachable states and reachable loops by symbolic techniques

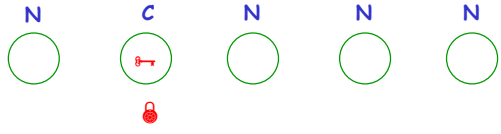
Potential applications to Word Structures

- Parameterized Systems with Linear or Ring Structure [Clarke,Grumberg,Jha], [Kesten,Maler,Pnueli,Shahar]
- Unbounded FIFO Channels [Boigelot,Godefroid,Wolper], [Bouajjani,Habermehl], [Abdulla,Bouajjani,Jonsson,Annichini]
- Pushdown Systems [Finkel,Willems,Wolper], [Bouajjani,Esparza,Maler], [Esparza,Schwoon], [Caucal], [Ball,Rajamani]
- Regular Hardware Structures [Basin,Klarlund et al],
- Systems with Integers and Reals [Boigelot,Wolper]
- Turing Machines

Implemented Tools
- MONA, LASH, TLV, PAX, …

Linear Process networks: Token passing

Action: Enter Critical Section



Token passing: Formal Model

Alphabet: $\Sigma = \{N , T , C\}$
Configuration: word over Σ
Initial Configurations: $T\ N^*$ (regular set over Σ)
Transition: pair of equally long words
e.g., N N T N N → N N N T N

Token passing: Formal Model

Alphabet: $\Sigma = \{N , T , C\}$
Configuration: word over Σ
Initial Configurations: $T\ N^*$ (regular set over Σ)
Transition: pair of equally long words
e.g., N N T N N → N N N T N
represent as sequence of pairs: ⟨N,N⟩ ⟨N,N⟩ ⟨T,N⟩ ⟨N,T⟩ ⟨N,N⟩

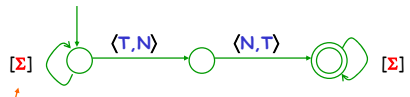
Token passing: Formal Model

Alphabet: $\Sigma = \{N , T , C\}$
Configuration: word over Σ
Initial Configurations: $T\ N^*$ (regular set over Σ)
Transition: pair of equally long words
e.g., N N T N N → N N N T N
represent as sequence of pairs: ⟨N,N⟩ ⟨N,N⟩ ⟨T,N⟩ ⟨N,T⟩ ⟨N,N⟩
Transition Relation: length-preserving relation on Σ^* represented by regular language over $\Sigma\times\Sigma$:

⟨N,N⟩* ⟨T,N⟩ ⟨N,T⟩ ⟨N,N⟩* is $\{N^{i+1}\ T\ N^j \rightarrow N^i\ T\ N^{j+1} ,\ i,j \geq 0\}$

Representing Actions (transition relations)

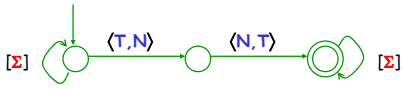
As transducers (finite automata over $\Sigma\times\Sigma$): [Kesten+ 97]



Notation:
[S] denotes $\{\langle a,a\rangle : a \in S\}$ i.e., "copy any symbol in S"

Representing Actions (transition relations)

As transducers (finite automata over $\Sigma\times\Sigma$): [Kesten+ 97]



As regular expressions over $\Sigma\times\Sigma$:

$[\Sigma]^*$ ⟨T,N⟩ ⟨N,T⟩ $[\Sigma]^*$

Notation:
[S] denotes $\{\langle a,a\rangle : a \in S\}$ i.e., "copy any symbol in S"

Representing Actions in MSO over strings

- Alphabet encoded by (typically boolean) configuration variables
- x[i] is the value of x at position i

Actions of the Token Ring Example

Pass token to right: $[\Sigma]^* \langle T,N\rangle \langle N,T\rangle [\Sigma]^*$

Pass token around(if ring): $\langle N,T\rangle [\Sigma]^* \langle T,N\rangle$

Enter Critical Section: $[\Sigma]^* \langle T,C\rangle [\Sigma]^*$

Exit Critical Section: $[\Sigma]^* \langle C,T\rangle [\Sigma]^*$

Idle: $[\Sigma]^*$

Verification Problems

Safety properties reduce to reachability

Mutual Exclusion:

- Is some configuration in $\Sigma^* C \Sigma^* C \Sigma^*$ reachable?

Verification Problems

Safety properties reduce to reachability

Mutual Exclusion:

- Is some configuration in $\Sigma^* C \Sigma^* C \Sigma^*$ reachable?

Liveness properties

- Token will reach righmost position:

  T N* **leads to** N*T

Verification Problems

Safety properties reduce to reachability

Mutual Exclusion:

- Is some configuration in $\Sigma^* C \Sigma^* C \Sigma^*$ reachable?

Liveness properties

- Token will reach righmost position:

  T N* **leads to** N*T

  needs specification of fairness properties

  e.g., **infinitely often** Pass-Token

Szymanski's Algorithm (idealized)

Mutual Exclusion algorithm for linear network of processes, indexed 1, … ,n

Pseudocode for process i

```
1: await ∀j : j ≠ i :: ¬s_j
2: w_i , s_i := true,true
3: if ∃j : j ≠ i :: (pc_j ≠ 1  /\ ¬w_j)
        then s_i := false; goto 4
        else  w_i := false; goto 5
4: await ∃j : j ≠ i :: (s_j /\ ¬w_j)
        then w_i , s_i := false,true
5: await ∀j : j ≠ i :: ¬w_j
6: await ∀j : j < i :: ¬s_j
7: s_i := false; goto 1
```


Szymanski's Algorithm (idealized)

Mutual Exclusion algorithm for linear network of processes, indexed 1, … ,n

Pseudocode for process i

```
1: await ∀j : j ≠ i :: ¬s_j
2: w_i , s_i := true,true
3: if ∃j : j ≠ i :: (pc_j ≠ 1  /\ ¬w_j)
        then s_i := false; goto 4
        else  w_i := false; goto 5
4: await ∃j : j ≠ i :: (s_j /\ ¬w_j)
        then w_i , s_i := false,true
5: await ∀j : j ≠ i :: ¬w_j
6: await ∀j : j < i :: ¬s_j
7: s_i := false; goto 1
```

"synchronized" moves

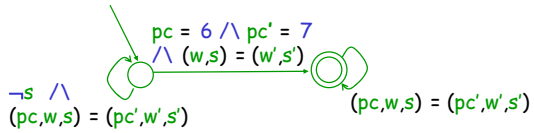
critical section

Encoding Szymanski's Algorithm

Alphabet: {(pc,w,s) : pc∈{1,2, … ,7} , w,s∈{0,1}}
(encoded in 5 bits)
Initially : (1,0,0)*
Transition relation: union of transducer for each action

Szymanski's Algorithm: Statement 6:

6: await $\forall j : j < i :: \neg s_j$



left context transformation right context

Verification Problems for Szymanski

Mutual Exclusion:
- Is some configuration in
  Σ* pc=7 Σ* pc=7 Σ*
  reachable?

In token ring, can be checked by backward reachability:
- Token ring is (almost) well-structured system
  - (actions are local) [Abdulla,Cerans,Jonsson,Tsay 96]
- Σ* C Σ* C Σ* is an upward closed set

Szymanski is not well-structured
- (actions have global conditions)

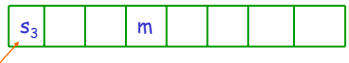
Verification Problems for Szymanski

Mutual Exclusion:
- Is some configuration in
  Σ* pc=7 Σ* pc=7 Σ*
  reachable?

Non-starvation:
More about this later …

Systems operating on numbers

- Sequence numbers, e.g., in communication protocols
  - Represented in "unary", i.e, as an array of states, msgs.
  - Typical configuration:

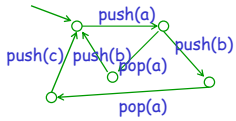


Control state of sender s_3, Next sequence number is 0

Channel contains message with sequence number 3

Representing Pushdown Systems

Control Part



Stack

Stack symbols from finite alphabet M

Alphabet: S ∪ M ∪ {⊥}
(S: control states; M: stack symbols; ⊥: empty slot)

Typical configuration: $\perp \perp \perp \perp \perp s_5\ m_1\ m_2\ m_5\ m_7$

Initial configurations: $\perp^* s_0$
(empty stack)

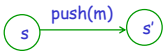
Representing Pushdown Systems

Alphabet: $S \cup M \cup \{\bot\}$

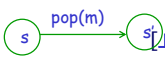
Typical configuration: $\bot \bot \bot \bot \bot s_5 \; m_1 m_2 m_5 m_7$

Initial States: $\bot^* s_0$

Actions:



$[\bot]^* \langle \bot, s' \rangle \langle s, m \rangle [M]^*$



$[\bot]^* \langle s, \bot \rangle \langle m, s' \rangle [M]^*$

Verification Algorithms

- Assume model with
  - Set of initial states Init
  - Transition relation Trans
- How to compute
  - Reachable configurations Init • Trans*
    Post* (Trans,Init)
  - Transitive closure Trans* of Trans (or of some other transition relation) for finding loops

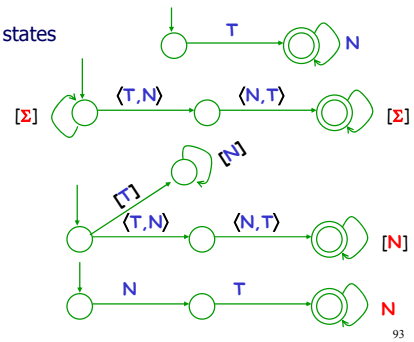
Using Automata for Reachability

Set of Initial states

Pass token

Product

Project and Minimize



What about Termination?

Set of initial states: $T N^*$

After exploring pass-action $T N^* \cup N T N^*$

After exploring k pass-actions $\bigcup_{i \le k} N^i T N^*$

How find the limit $N^* T N^*$ **?**

Compute post*(pass-token, $T N^*$) = $N^* T N^*$
by Acceleration/Widening.

For Transitive Closure

Pass action : $[\Sigma]^* \langle T,N \rangle \langle N,T \rangle [\Sigma]^*$

Transitive Closure not regular

Restrict by Reachable states : $[N]^* \langle T,N \rangle \langle N,T \rangle [N]^*$

Composing at most k pass-actions

$\bigcup_{i \le k} [N]^* \langle T,N \rangle [N]^{i-1} \langle N,T \rangle [N]^*$

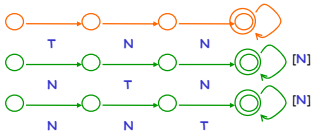
How find the limit $[N]^* \langle T,N \rangle [N]^* \langle N,T \rangle [N]^*$ **?**

Verification Techniques and Results

- Rules for actions with certain forms [Abdulla, Jonsson, Nilsson CAV99][Fisman Pnueli Shahar, CAV00]
- Decidability result for systems with "bounded local depth" [Jonsson,Nilsson,TACAS00]
- Extrapolation [Bouajjani,Jonsson,Nilsson,Touili,CAV00, Touili VEPAS 01]
- Techniques based on merging states in transducer [Dams,Lakhnech,Steffen,CAV01] [Abdulla,Jonsson,Nilsson,d'Orso,Saksena,CONCUR02,CAV03]
- And more…

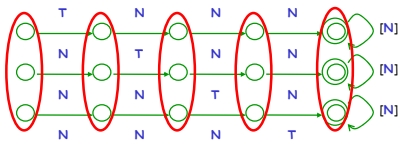
Acceleration for Token Passing

Init ∘ Pass-Action^2



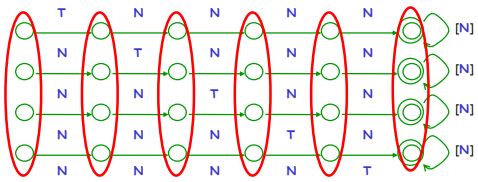
Acceleration for Token Passing

Pass-Action^3



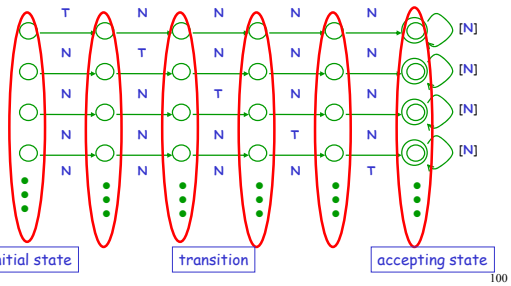
Acceleration for Token Passing

Pass-Action^4



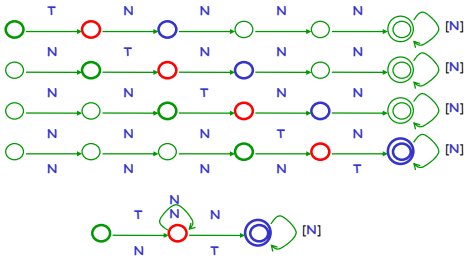
Column Transducer

Pass-Action$^{\omega}$ = U_k Pass-Action^k



Acceleration for Token Passing

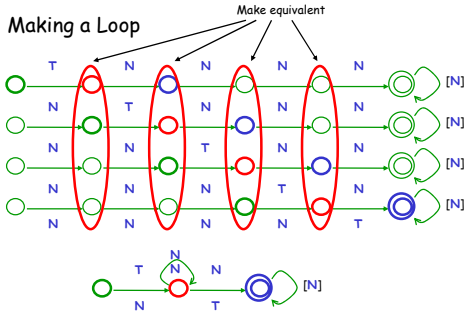
Making a Loop



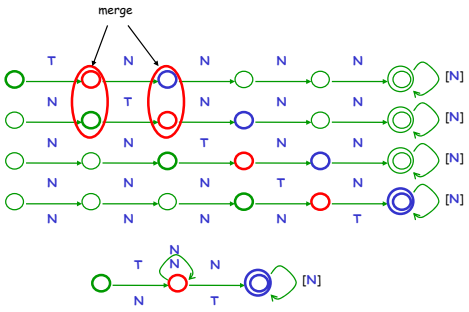
Acceleration for Token Passing

Making a Loop

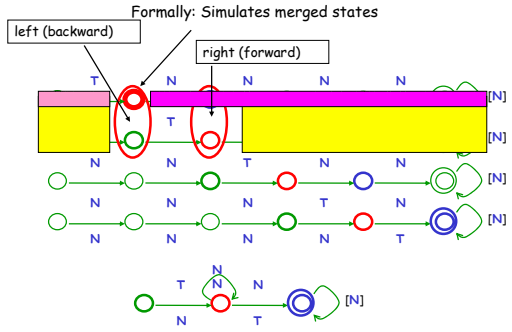
Make equivalent



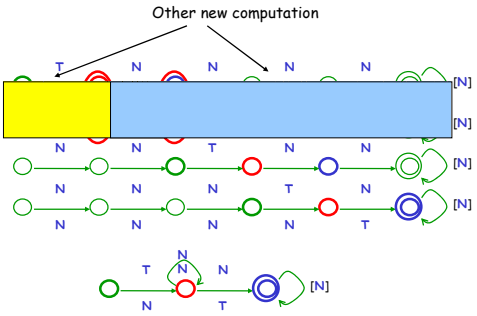
Acceleration for Token Passing



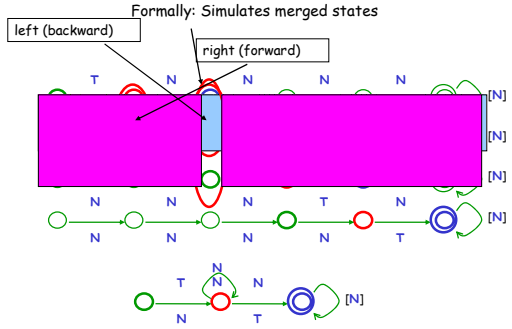
Acceleration for Token Passing



Acceleration for Token Passing

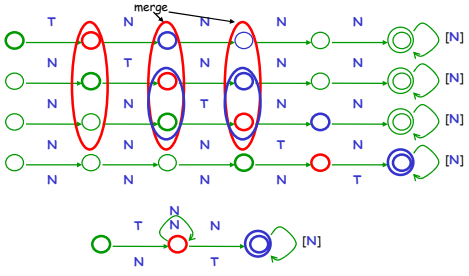


Acceleration for Token Passing



Acceleration for Token Passing

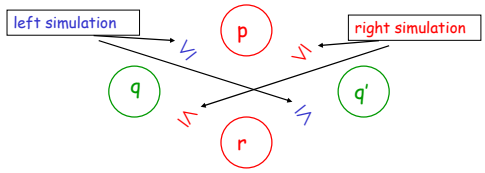
If simulations are congruences, the trick can be repeated for more loop iterations



MergingTheorem

[Dams,Lakhnech,Steffen01][Abdulla et al 03]

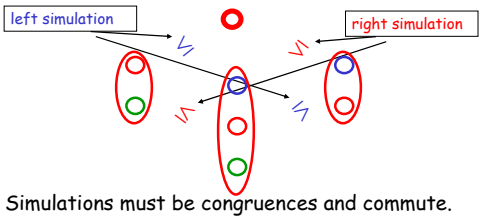
Merge states q and q' if there are dominating p and r



Simulations must be congruences and commute.

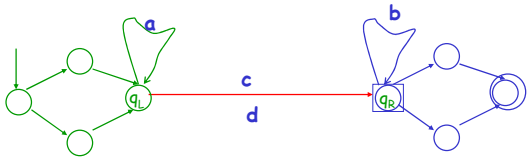
]Application to Token Passing

Merge states q and q' if there are dominating p and r



Simulations must be congruences and commute.

Special case: Unary actions



Left Context: Copying, forward deterministic

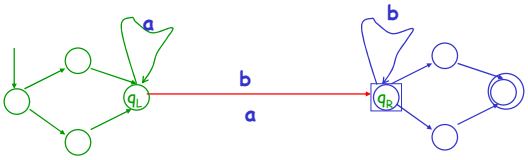
Noncopying states

Right Context Copying, backward deterministic

Complete characterization for c ≠ d

[Abdulla Jonsson NilssonCAV99]

Special case: Unary actions

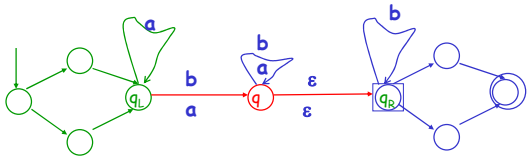


Left Context: Copying, forward deterministic

Noncopying states

Right Context Copying, backward deterministic

Limit $Trans^+$



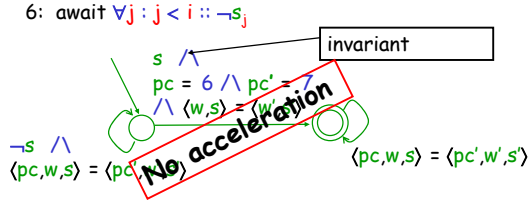
Left Context: Copying, forward deterministic

Noncopying states

Right Context Copying, backward deterministic

Szymanski's Algorithm: Statement 6:

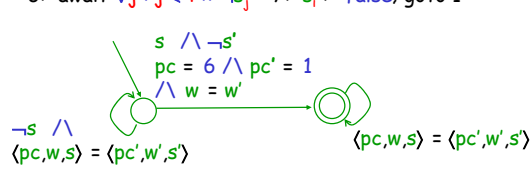
6: await $\forall j : j < i :: \neg s_j$



left context transformation right context

Szymanski's Algorithm: Statement 6+7:

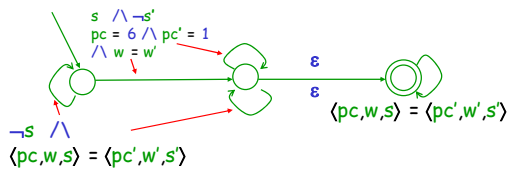
6: await $\forall j : j < i :: \neg s_j$ 7: s_i := false; goto 1



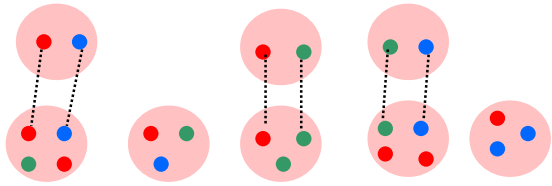
left context transformation right context

Acceleration of Statement 6+7:

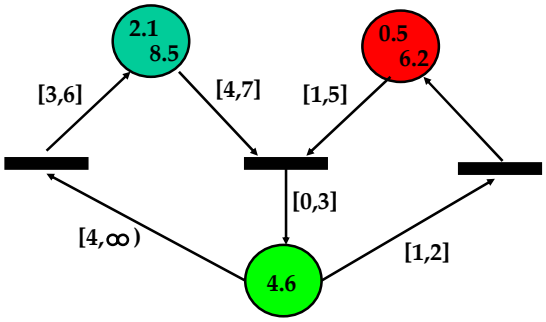
6: await $\forall j : j < i :: \neg s_j$ 7: s_i := false; goto 1



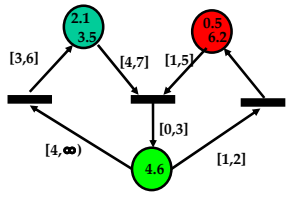
Words of multisets over a finite alphabet



Timed Petri Nets



States = Markings



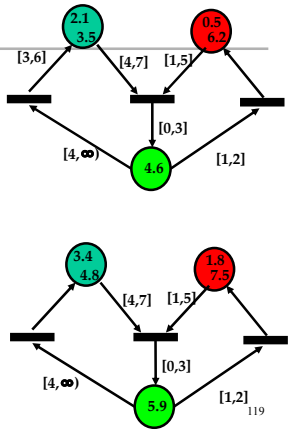
2.1 3.5 0.5 6.2 4.6

Timed Transitions

2.1 3.5 0.5 6.2 4.6

increase age by 1.3

3.4 4.8 1.8 7.5 5.9

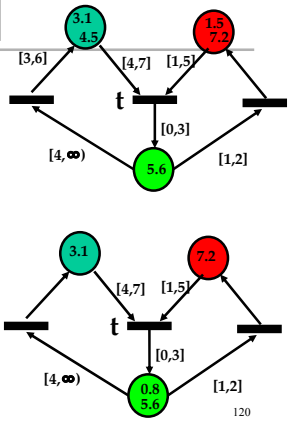


Discrete Transitions

3.1 4.5 1.5 7.2 5.6

Firing t

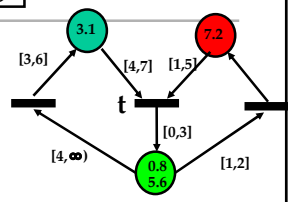
3.1 7.2 0.8 5.6



Timed Petri Nets

- Concurrent timed systems
- Infinite-state: symbolic representation
- Monotonic behaviour
- Safety properties: reachability of ideals

Equivalence on Markings



- max = 7
- ages > max behave identically

Equivalence on Markings

Markings equivalent if they agree on:
- colours
- integral parts of clock values
- ordering on fractional parts

3.1 4.8 1.5 6.2 5.6 → 3.1 1.5 4.8

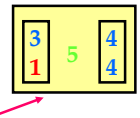
3.2 4.8 1.6 6.4 5.7 ⋯→ 3.2 1.6 4.7

Equivalence on Markings

Markings equivalent if they agree on:
- colours
- integral parts of clock values
- ordering on fractional parts

3.1 4.8 4.8 1.1 5.4

3.2 4.7 4.7 1.2 5.5



words over multisets over a finite alphabet

Ordering on Markings

$M_1 \sqsubseteq M_2$ iff $\exists M_3$:
- $M_1 \equiv M_3$
- $M_3 \leqslant M_2$

4.8 6.4 5.7 ⊓ 3.1 4.8 1.5 6.2 5.6 ≡ 4.8 6.2 5.6

4.8 6.4 5.7 ⊓ 3.1 4.8 1.5 6.2 5.6 ≡ 4.8 6.2 5.6

6 5 4 = 6 5 4

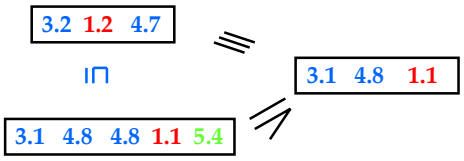
subword

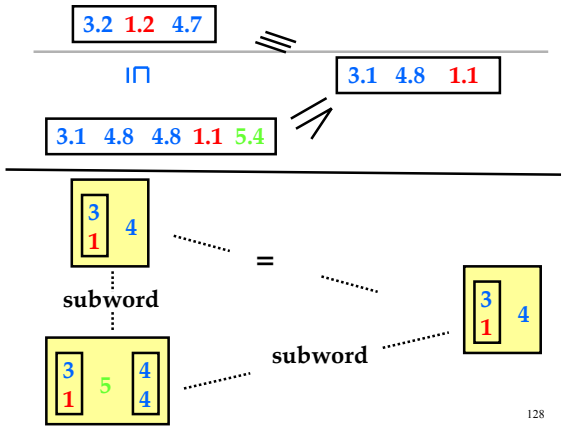
3 6 1 5 4 subword

Ordering on Markings

$M_1 \sqsubseteq M_2$ iff $\exists\, M_3$:

- $M_1 \equiv M_3$
- $M_3 \leq M_2$



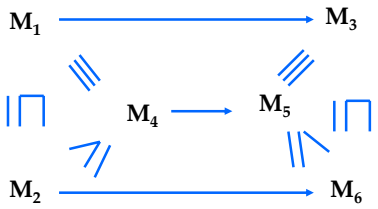


Properties of $\sqsubseteq$

$\sqsubseteq$ = subword ordering on multisets over a finite alphabet

$\sqsubseteq$ is a well quasi-ordering

Properties of $\sqsubseteq$ -- Monotonicity



Methodology (applied to TPN)

1. Computable ordering
2. Monotonicity, Computability of Pre
3. Ordering is WQO

There are many other well-structured systems.

- Systems with integer variables and monotone operations
  - Petri Nets, Broadcast Protocols [Bouajjani Esparza Mayr]
- Timed Automata,
- Relational Automata [Cerans]
- Finite State systems with Lossy FIFO Channels
- Timed Petri Nets (without urgency)