Abschnitt I

Asynchronous System Model

Inhalt: I/O automata · traces and executions · operations on automata: composition + hiding · fairness · properties and proof methods · safety and liveness

Literatur: The material is taken from [Lyn96, Chapter 8].

Overview

- big step: from synchronous to asynchronous[1] model
- more complex
  - more "nondeterminism"
  - more uncertainty, due to the relative speed of the parallel components
  - fairness, liveness
- machine model:

$$\boxed{\text{I/O automata}}$$

[1] the word "asynchronous" is used differently sometimes elsewhere. *Asynchronous* communication/message passing is often meant as *buffered* communication.

I/O automaton

- simple machine/automaton model for processes in an asynchronous distributed network
- named "actions" for the transitions (internal or external = input or output)
- $\Rightarrow$ notion of interface/signature
- composable
- general/unspecific enough for
  - shared memory concurrent systems (cf. Chapter 9 and following)
  - message passing ("network") systems (cf. Chapter 14 and following)
- Examples:

1. process P_i of Figure 8.1
2. fifo channel $C_{i,j}$ of Figure 8.2[2]

[2] One sees that the model is so unspecific, that channels are not built in, but have to be programmed.

Actions and signature

- actions = will be "labels" on transition
- signature $S = (in(S), out(S), int(S))$: three disjoint set of actions:
  1. $int(S)$: internal
  2. $out(S)$: output
  3. $in(S)$: input
- furthermore
  - external actions: $ext(S) \triangleq in(S) \dot{\cup} out(S)$
  - locally controlled actions $local(S) \triangleq int(S) \dot{\cup} out(S)$.
  - thus: external signature/interface $extsig(S) = (in(S), out(S), \emptyset)$
  - A closed if $in(A) = \emptyset$ ("autonomous")

I/O automaton: Definition

Definition 1. *An I/O automaton A is given by*

1. $sig(A)$, *a signature*
2. $states(A)$ *(finite or countably infinite)*
3. $start(A)$*: subset of initial states*
4. $trans(A) \subseteq states(A) \times acts(sig(A)) \times states(A)$*: state-transition relation, input-enabled*
5. $tasks(A)$*: task-partition* $=$ *equivalence-relation on* $local(sig(A))$*, at most countably infinite equivalence classes*

Enabledness, tasks

- transition/step: $(s, \pi, s') \in trans(A)$[3] (input/output/internal . . . transitions)
- action π enabled in s, if $s \xrightarrow{\pi} s' \in trans(A)$.
- input-enabledness: every input action must be enabled in every state
  - it's better to consider all possible reaction, otherwise: error prone design
  - nicer theory
- quiescent state: no actions except input actions are enabled[4]
- tasks
  - abstract represention for "tasks/jobs/threads of control"
  - useful specifically for "parallel composition"
  - primarily used later to specify fairness conditions to ensure liveness properties
  - word "task" $\hat{=}$ task-partition class of the automaton

[3] I sometimes write also $s \xrightarrow{\pi}_A s'$, or $s \xrightarrow{\pi} s'$, if A is clear from the context.
[4] Cf. the definition of quiescent state for the synchronous model [Lyn96, p. 19].

Examples: channel and process

- cf. Examples 8.1.1 and 8.1.2 in [Lyn96, p. 204]
- written in precondition/effect-style, grouping together "analogous" actions (parameter)
- note:
  - input enabledness: empty precondition = true
  - *tasks*:
    - for the process: separation per receiver channel
    - for the channel: all outputs[5] in the same class

[5] i.e., all locally controlled actios in this example.

Executions and traces

- straightforward step semantics
  - starting from an initial state: do transitions
  - non-deterministic[6]
  - we distinguish between
    * internal steps observable: executions
    * only actions of external steps observable: traces $=$ interface behavior

Definition 2. [Traces and executions] *Given A*

1. an execution fragment of A is

(a) finite sequence $s_0\pi_1 s_1 \pi_2 \ldots \pi_r s_r$
(b) or infinite sequence $s_0\pi_1 s_1 \pi_2 \ldots$,

such that $s_i \stackrel{\pi_{i+1}}{\rightarrow} s_{i+1}$. Note: in the finite case we end in a state.

[6] because the transitions allow this. Parallel composition will add another source of non-determinism.

execution = *execution fragment starting with an initial state*

2. *traces:*

   (a) the *trace of an execution (fragment)* α *of A: subsequence of* α *consisting of the external actions*[7]

   (b) *of an automaton* A*: lifted on the set* A*'s executions*

- notation for executions and traces: $execs(A)$, $trace(\alpha)$, $traces(A)$.
- a state s is reachable, if there exists a (finite) execution with s and end-state
- concatenation: $\alpha \cdot \alpha' =$ "glueing" together (finite) execution fragments[8]
- Example 8.1.3 about the Fifo-channel $C_{i,j}$

[7] no states!

[8] Assuming, that the end state of α equals the first state of α' and of course, not mention the glue state twice in $\alpha \cdot \alpha'$, ...

Composition

- parallel composition of larger/more complex system by smaller ones
- hierarchical description
- standard product construction:
  - states are paired
  - transitions ("interleaving")
    - synchronizing on common actions
    - non-synchronizing on local actions: automaton does nothing[9]
- to obtain the desired intuition: certain restrictions
  - internal actions should not synchronize: internal actions disjoint[10]

[9] In other models, one could call this *stuttering* steps.

[10] That's one possible way to formalize the informal, intuitive intention that some actions are considered to be internal.

– only one process controls other actions:[11] output actions disjoint[12]
– only a *finite* number of synchronizing, common actions

⇒ definition of compatible signatures and automata

[11] By synchronizing over output actions. It's one of the underlying intuitions in this model: input forces the component to do someting and especially the component cannot *refuse* to accept input, but the internal actions and the output actions are under *component control.*

[12] Note that input actions are not required to be disjoint: an "outputter" can trigger many "inputters". This form of multi-synchronization is allowed.

Compatible

a countable collection $\{S_i\}_{i \in I}$ is compatible:

locality $int(S_i) \cap acts(S_j) = \emptyset$

indepedent outputs $out(S_i) \cap out(S_j) = \emptyset$

finite sync. no action is contained in infinitely many $acts(S_i)$

- note: $local(S_i) \cap local(S_j) = \emptyset$
- compatible collection of automata: correspondingly

$\Rightarrow$ composition of sig's and automata

Composition: signatures

signatures: given a compatible, countable collection $\{S_i\}_{i \in I}$ of signatures

$\Rightarrow$ $\boxed{S = \prod_{i \in I} S_i}$ given by

- $out(S) \triangleq \dot{\bigcup}_{i \in I} out(S_i)$
- $int(S) \triangleq \dot{\bigcup}_{i \in I} int(S_i)$
- $in(S) \triangleq \bigcup_{i \in I} in(S_i) - \dot{\bigcup}_{i \in I} out(S_i)$

Composition: automata

automata : given a compatible, countable collection $\{A_i\}_{i \in I}$ of automata

$\Rightarrow$ $\boxed{A = \prod_{i \in I} A_i}$ given by

- $sig(A) \triangleq \prod_{i \in I} sig(A_i)$
- $states(A) \triangleq \prod_{i \in I} states(A_i)$
- $start(A) \triangleq \prod_{i \in I} start(A_i)$
- $s \xrightarrow{\pi} s' \in trans(A)$ if, for all i
  - if $\pi \in acts(A_i)$: $s_i \xrightarrow{\pi}_i s'_i$
  - if $\pi \notin acts(A_i)$: $s_i = s'_i$
- $tasks(A) \triangleq \bigcup_{i \in I} tasks(A_i)$.

Composition: remarks

- remember: A_i's are input enabled $\Rightarrow$ A is, too
- note: local actions cannot be used to synchronize ($=$ more than one process makes a real step) by convention
- for finite/binary composition: $A \times B$, or $A_1 \times \ldots \times A_n$
- it's to prove, that $\prod_{i \in I} A_i$ yields an automaton, same for signatures
- note: composing output and input actions —both are external— yields: an output[13] $\Rightarrow$ broadcast communication can directly be modelled.
- $\times$ is associative[14]

[13]Other known models make a different plausible choice here: input parallel with an output gives an internal action.

[14]Technical remark: associativity hinges, in this formalization of parallel composition on the fact: output $+$ input gives output, not an internal action. This, on the other hand, does not mean to say, that *binary* associative synchronization is impossible. One would have to give up to *force* synchronization on common actions in parallel actions. Such a communication model is, for instance, used in CCS, and similar calculi.

Composition: example

- paralell composition of processes + buffers: 8.2.1

Local and global executions/traces

- relating executions of a composed automaton $\leftrightarrow$ executions of the components
- given $\alpha = s_0\pi_1 s_1 \ldots$ of A: "projection" $\alpha \downarrow_{A_i}$ = subsequence obtained by
  1. deleting all pairs $\pi_r s_r$ where π_r not an A-action
  2. replace each remaining s_r by the i-th component s_i
- projection is analogously used on traces or arbitrary sequences of actions

Theorem 1. [Decomposition (8.1)] *Given* $A = \prod_{i \in I} A_i$.

- $\alpha \in execs(A)$, *then* $\alpha \downarrow_{A_i} \in execs(A_i)$
- $\alpha \in traces(A)$, *then* $\alpha \downarrow_{A_i} \in traces(A_i)$

Theorem 2. [Composition from executions (8.2)] *Given*

- $A = \prod_{i \in I} A_i$
- α_i: *an execution of* A_i
- β: *a sequence of actions in* $ext(A)$ *s.t.* $\beta \downarrow_{A_i} = trace(\alpha_i)$

Then there is an execution α *of* A *such that* $\beta = trace(\alpha)$ *and* $\alpha_i = \alpha \downarrow_{A_i}$

Theorem 3. [Composition from traces (8.3)] *Given*

- $A = \prod_{i \in I} A_i$
- β *a sequence of actions in* $ext(A)$
- $\beta \downarrow_{A_i} \in traces(A_i)$

Then

$$\beta \in traces(A).$$

Hiding

- operation on an I/O-automaton: hiding output
- after hiding: operation is internal $\Rightarrow$ no further synchronization possible

for signatures: given S and $\Phi \subseteq out(S) \Rightarrow hide_\Phi(S) = S'$ defined as

- $int(S') \triangleq int(S) \cup \Phi$
- $in(S') \triangleq in(S)$
- $out(S') \triangleq out(S) - \Phi$

note: $\Phi \cap in(S') = \emptyset$ by definition

for automata: simply using the definition for signatures, i.e., given A and $\Phi \subseteq out(A)$:

- $hide_\Phi(A)$ defined as A' given by replacing $sig(A)$ by $hide_\phi(A)$

Fairness: informal

- asynchronous model: fairness becomes quite an issue
- informally: fair = "each one get's his turn" (here based on tasks)
- formally: abstract (and often difficult) notion, considering infinite behavior
- various "flavors" of fairness useful
- abstraction of a scheduler
- here: each task gets infinitely many opportunities to perform one of its actions

Fairness

Definition 3. [Fairness] *An execution fragment α of A is fair, if for each equivalence class C of $tasks(A)$:*

1. if α is finite, then C is not enabled in the final state of α

2. if α is infinite, then α contains

(a) infinitely many events from C, or
(b) infinitely many occurences of states in which C is disabled

- event $=$ occurrence of an action in a sequence (execution, trace)
- $fairexecs(A)$: fair executions of A, and $fairtraces(A)$: fair traces of A, where a fair trace is a trace of a fair executions[15]

[15] Remember: trace cf. slide 9.

- example 8.3.1

- example 8.3.2: *discrete clock*

(De-)composition of fair traces

Analogous to the corresponding properties for general traces/executions

Theorem 4. [Fair decomposition (8.4)] *Given* $A = \prod_{i \in I} A_i$*.*

- $\alpha \in \mathit{fairexecs}(A)$, *then* $\alpha \downarrow_{A_i} \in \mathit{fairexecs}(A_i)$
- $\alpha \in \mathit{fairtraces}(A)$, *then* $\alpha \downarrow_{A_i} \in \mathit{fairtraces}(A_i)$

Theorem 5. [Composition from fair executions (8.5)] *Given*

- $A = \prod_{i \in I} A_i$
- α_i*: a fair execution of* A_i
- β*: a sequence of actions in* $\mathit{ext}(A)$ *s.t.* $\beta \downarrow_{A_i} = \mathit{trace}(\alpha_i)$

Then there is a fair execution α *of* A *such that* $\beta = trace(\alpha)$ *and* $\alpha_i = \alpha \downarrow_{A_i}$

Theorem 6. [Composition from fair traces (8.6)] *Given*

- $A = \prod_{i \in I} A_i$
- β *a sequence of actions in* $ext(A)$
- $\beta \downarrow_{A_i} \in fairtraces(A_i)$

Then

$$\beta \in fairtraces(A).$$

Fairness example

- Example 8.3.3: 3 processes
  - every sent message is eventually received
  - if there is at least one init-event for each i, each processes sends infinitely many messages to each other, and each process send infinitely many decide messages

Fairness and finite prefix

"(Un-)fairness does not show up on finite prefixes of executions"

- fairness is not a safety property
- more formally: every finite executions can be extended into a fair execution (same for traces)

Theorem 7. [Fairness] *1. If α is a finite execution of A, then there is a fair execution of A that starts with α.*

2. The same holds analogously for traces.

3. If α is a finite execution of A and β is any (finite or infinite) sequence if input actions of A, then there is a fair execution $\alpha \cdot \alpha'$ of A such that the sequence of input actions in α' is exactly β.

4. *similar for traces: If* β *is a finite trace of* A *and* β' *is any (finite or infinite) sequence if input actions of* A*, then there is a fair execution* $\alpha \cdot \alpha'$ *of* A *such that* $trace(\alpha) = \beta$ *and s.t. the sequence of input actions in* α' *is exactly* β'*.*

Input and output

- to apply this general model: make some conventions about I/O
- remember: in the synchronous model: designated state variables (write-once for outputs)
- in the asynchronous model: we simply use input and output actions

Invariant assertions

- most basic/simple class of properties: invariant assertion or invariant

$\boxed{\text{“something which always holds”}}$

- invariant of an automaton A: property which holds for all reachable states
- typically proven by induction on the number of steps[16]

[16] In the synchronous setting, we used induction on the number of rounds.

Trace properties and satisfaction

- remember: trace = "visible/observable" actions of an execution

$\Rightarrow$ often, properties of "external" interest are formulated over (fair) traces, i.e., extensionally by a set of traces,

Trace property P given by

1. $sig(P)$, signature, containing no internal action[17]
2. $traces(P) \subseteq acts(sig(P))$

- (at least) 2 interpretations of A satisfies P ("$A \models P$")

1. $extsig(A) = sig(P)$ and $traces(A) \subseteq traces(P)$
2. $extsig(A) = sig(P)$ and $fairtraces(A) \subseteq traces(P)$

[17] One cannot "look inside"

Safety

- important class of trace properties[18]
- general slogan

> "never something bad happens"

Definition 4. [Safety trace property] *A trace property P is a safety (trace) property:*

1. $traces(P)$ is not empty

2. $traces(P)$ is prefix closed[19]

[18] In this setting, safety is phrased in terms of trace properties.
[19] That implies the first point already.

3. $traces(P)$ *is limit closed*

Definition 5. [Limit closure] *An set of traces* T *is limit closed: if* $\beta_1, \beta_2, \ldots$ *is an infinite sequence of finite traces in* T*, such than for all* i*:*

$$\beta_i \text{ is a prefix of } \beta_{i+i} \ ,$$

then the limit β *(the unique sequence* β *that is the limit of* β_i *under the sucessive extension ordering) is also in* T*.*

- intuition: if something bad happens, it happens in a finite amount of time by a particular event $\Rightarrow$ limit closure

- note: a fairness property is not a safety property

Example 1. [8.5.2] *"No* $decide$ *happens without a preceding* $init$*".*

Lemma 1. *If* P *is a safety property, then the following statements are equivalent*

1. $traces(A) \subseteq traces(P)$*.*

2. $fairtraces(A) \subseteq traces(P)$.

3. $finitetraces(A) \subseteq traces(P)$.

Liveness

- general slogan:

$\boxed{\text{“something good will (indeed) happen”}}$

Definition 6. [Liveness] *A trace property P is a (trace) liveness property, if every finite sequence over $acts(P)$ has some extension in P.*

Example 2. [8.5.3] *For every $init$” event, there will be a/infinite many matching $decide$ later.*

Proving liveness/temporal logic(s)

- proving a liveness property:[20] $\mathit{fairtraces}(P) \subseteq \mathit{traces}(P)$ = liveness property
- temporal logic:
  - logic(s) targeted towards reasoning about reactive behavior/traces/ ...
  - "temporal" does not (necessarily) mean real time: but ordering of events
  - typical operators: "Always", "Eventually" (sometimes written $\Box$, $\Diamond$, $\bigcirc$ ...)
- "do-it-yourself-method": progress functions[21]
  - proving that a particular event will happen
  - mapping from states of the automaton tp well-founded set
  - show that actions decrease the value

[20] Often, fairness assumptions needed to prove liveness.

[21] one can use temp. logic to formalize this method. Anyway, we won't use temporal logic.

Safety and liveness

- safety and liveness: intuitive dual classes of properties
- this can be made formal/proven (in the following)
- note: the properties are phrased for traces, analogous properties holds for executions

Theorem 8. [Safe and live (8.8)] *P is a safety and a liveness property $\Rightarrow$*

$$P = acts(P)$$

Theorem 9. [Safety and liveness decomposition] *Given trace property P with $traces(P) \neq \emptyset$. Then there exist a safety and a liveness property S and L:*

1. $sig(S) = sig(L) = sig(P)$.
2. $traces(P) = traces(S) \cap traces(L)$.

Compositional reasoning

- “divide-and-conquer” approach sometimes helpful in reasoning
- given[22] $A = \prod_{i \in I} A_i$ and $P = \prod_{i \in I} P_i$, and local satisfaction: $A_i \models P_i$

Theorem 10. [Compositional reasoning] *1. If $extsig(A_i) = sig(P_i)$ and $traces(A_i) \subseteq traces(P_i)$, then and $extsig(A) = extsig(P)$ and $traces(A) \subseteq traces(P)$.*

2. If $extsig(A_i) = sig(P_i)$ and $traces(A_i) \subseteq fairtraces(P_i)$, then and $extsig(A) = extsig(P)$ and $traces(A) \subseteq fairtraces(P)$.

[22] Unless stated otherwise: by writing the products $\prod_{i \in I} A_i$ and $\prod_{i \in I} P_i$, we implicitly assert that those are well defined, especially that the constituents form a compatible collection of automata resp. trace properties.

$$\frac{extsig(A_i) = sig(P_i) \qquad traces(A_i) \subseteq traces(P_i)}{traces(A) \subseteq traces(P)}\ \textsc{Comp-Tr}$$

$$\frac{extsig(A_i) = sig(P_i) \qquad fairtraces(A_i) \subseteq traces(P_i)}{fairtraces(A) \subseteq traces(P)}\ \textsc{Comp-FTr}$$

Further compositional reasoning

- using composition Theorem 3 on page 19 (or the corresponding one for fair traces)
  - "inverting" the previous reasoning
  - drawing conclusion from projections to the common trace.
- compositional proof of safety properties

Compositional reasoning for safety properties

- exploiting that safety is violated in a finite prefix
- A preserves P: A is not the first to violate P
- if no A_i is the first violator, safety holds

Definition 7. [Preserving] *P a safety property with $acts(P) \cap int(A) = \emptyset$ and $in(P) \cap out(A) = \emptyset$.*

A preserves B, if for every finite sequence β of actions that does not include internal actions of A, and every $\pi \in out(A)$:

> *If $\beta \downarrow_{acts(P)} \in traces(P)$ and $\beta\pi \downarrow_A \in traces(A)$, then $important \beta\pi \downarrow_{acts(P)} \in traces(P)$.*

The internal actions are not mentioned, of course.

Compositional reasoning & safety

Theorem 11. [Safety and comp. reasoning] *Given* $A \ \prod_{i \in I} A_i$. *and a safety property with* $acts(P) \cap int(A) = \emptyset$ *and* $in(P) \cap out(A) = \emptyset$*:*

1. *If* A_i *preserves* P *for all* i*, then* A *preserves* P

2. *If* A *is closed,* A *preserves* P*, and* $acts(P) \subseteq ext(A)$*, then*

$$traces(A) \downarrow_{acts(P)} \subseteq traces(P) \ .$$

3. *If* A *is closed,* A *preserves* P*, and* $acts(P) = ext(A)$*, then*

$$traces(A) \subseteq traces(P) \ .$$

Hierarchical proofs

- hierarchy: different level of abstraction
- from top to bottom:[23] successive refinements
- examples: simulation proofs in the material of the synchronous model
- problem: the two systems under comparison are not a strictly coupled as before
- needed: generalization of the simulation method to the asynchronous setting
- anyway: general approach: directed relationship:[24]

  "for any execution of the lower-level automaton, there is a "corresponding" execution of the higher-level automaton."

[23] "Higher" means: more abstract, less details etc. One could for instance intruduce more parallelism going top-down.

[24] In the synchronous setting, the goal was that the implementation had the same behavior than the abstract system., at least under the assumption of determinism.

Simulation relation

Definition 8. [Simulation relation] • *A and B with identical external interfaces.*

- $f \subseteq states(A) \times states(B)$.

Then f is a simulation relation from A to B, if

Start condition: *if $s \in start(A)$, then $f(s) \cap start(B) \neq \emptyset$*

Step condition: • *if s is a reachable state of A, and if $u \in f(s)$ where u is a reachable state of B:*

- *$s \xrightarrow{\pi} s'$, then there is an execution fragment α of B, starting in u and ending with some $u' \in f(s')$ such that*

$$trace(\alpha) = trace(\pi)$$

Simulation proof method

- important proof technique, e.g., for trace
- aux. definition: B simulates A ($B \succeq A$), if there exists a simulation relation from A to B.

  alternative words: A is simulated by B ($A \preceq B$)
- Caveat: some people use the words “the other way around”

Theorem 12. [Simulation and trace inclusion] *If there is a simulation relation from A to B, the the traces of A are included in the traces of B:*

$$\frac{A \preceq B}{\mathit{traces}(A) \subseteq \mathit{traces}(B)}\ \textsc{Sim}$$

Simulation and safety property

- simulation: preservation of safety properties[25]

$$\frac{P \text{ is a safety property} \qquad B \preceq A \qquad A \models P}{B \models P}$$

- for liveness properties:
  - not that simple/direct

[25] Remember: $A \models P$ means $traces(A) \subseteq traces(P)$.

Simulation and liveness properties

- we need to strengthen the coupling

Definition 9. [Correspondance] *Given:*

- A *and* B *with identical input and output actions*
- α *and* β *executions of* A *resp.* B
- *relation* $f \subseteq states(A) \times states(B)$.

Then α *and* β *correspond wrt.* f *(written* $\alpha \bowtie_f \beta$*), if*

1. *there exists a mapping* g *from indices (occurrences) of states in* α *to indices of states in* β*,*
   - g *is monotone nondecreasing*

- g *exhausts all of* β[26]
- g*-corresponding pairs of states are related by* f
- *between successive* g*-corresponding pairs of states, the traces in* α *and* β *are identical*

Theorem 13. [Simulation with correspondence] *If there is a simulation relation* f *from* A *to* B[27]*, then for every execution* α *of* A*, there exists execution* β *of* B *such that* $\alpha \bowtie_f \beta$.

[26] I.e., the suprenum of the range of g is the supremum of the indices of states in β.

[27] I.e., $A \preceq B$

Complexity measures

- upper time bound for any subset of equivalence classes in $tasks(A)$ (thus also for any task, a "full" equivalence class) $\Rightarrow$
- $upper_C \in \mathbb{R}^{>0} + \infty$
- given a fair execution α of $A \Rightarrow$ associate a real-value time with each event of α.
  1. times are monontone non-decreasing in α
  2. if α is infinite, then the times approach ∞[28]
  3. from any point in α, a task C can be enabled for time at most $upper_C$, before some action in C must occur
- timed execution $=$ fair execution with times associated as described
- note: for a given set of bounds in $upper_C$: many ways of associating times to the events of α

[28] sometime called non Zeno-ness.

- time until some designated event π in α occurs: supremum of times assignable to π in all such timed executions

- time between two events: likewise by the supremum of differences

- example 8.6.1

Indistinguishable executions

- indistinguishable $=$ from the perspective of a subcomponent $\Rightarrow$ projection

Definition 10. [Indistinguishable] *Given: α, α': executions of two composed systsm each containing automaton A. Then:*

α and α' are indistinguishable to A, if their projections onto A are identical:

$$\alpha \sim_A \alpha' \triangleq \alpha \downarrow_A = \alpha' \downarrow_A$$

Randomization

- remember: transition relation in the asynchronous model: nondeterministic[29]
- sometimes: "weighted" nondeterminism is useful ⇒ randomization/probabilistic I/O automata

Definition 11. [Probabilistic I/O automaton] *A probabilistic I/O automaton is defined the same way as an ordinary I/O automaton (cf. Definition 1 on page 6), except that the successor states are given by a probability distribution*

$$(s, \pi, P) ,$$

where P is a probability distribution over some subset of the states.

- semantics/execution: series of 2 non-deterministic choices!

[29] In the synchronous (network) model, we had some next-state *function*.

1. choose the next transition $s \xrightarrow{\pi} P$
2. choose the state according to the distribution

- restriction: choice of the transition should be fair
- "inside" each probablistic automaton, there is a nondeterministic one, forgetting the distribution ($\mathcal{N}(A)$)[30]

[30] sucessor states with probability 0 are not represented in the non-deterministic choice.

Literatur

[Lyn96] Nancy Lynch. *Distributed Algorithms*. Kaufmann Publishers, 1996.