Dipartimento di Informatica, Università di Pisa

Notes on

Models of Computation

Parts I-V

Introduction, Preliminaries
Operational Semantics of IMP
Induction, Recursion
Partial Orders, Denotational Semantics of IMP
Operational Semantics of HOFL, Domain Theory
Denotational Semantics of HOFL
CCS, Temporal Logic, μ-calculus, π-calculus
Markov Chains with Actions and Nondeterminism
PEPA

Roberto Bruni Lorenzo Galeotti Ugo Montanari*

May 22, 2014

*Also based on material by Andrea Cimino, Lorenzo Muti, Gianmarco Saba, Marco Stronati

Contents

Part III.

CCS

10. CCS, the Calculus for Communicating Systems

In the last decade computer science technologies have boosted the growth of large scale distributed and concurrent systems. Their formal study introduces several aspects which are not present in the case of sequential systems. In particular, it emerges the necessity to deal with non-determinism, parallelism, interaction and infinite behaviour. Non-determinism is needed to model time races between different signals and to abstract from programming details which are irrelevant for the interaction behaviour of systems. Parallelism allows agents to perform tasks independently. For our purposes, this will be modelled by using non-determinism. Interaction allows to describe the behaviour of the system from the observational point of view (i.e., the behaviour that the system shows to an external observer). Infinite behaviour allows to study the semantics of non-terminating processes useful in many different contexts (e.g., think about the modelling of operating systems). Accordingly, from the theoretical point of view, some additional efforts must be spent to extend the semantics of sequential systems to that of concurrent systems in a proper way.

As we saw in the previous chapters, the study of sequential programming languages brought to different semantics which allows to prove many different properties. In this chapter we introduce CCS, a specification language which allows to describe concurrent communicating systems. Such systems are composed of agents (i.e., processes) performing tasks by communicating each other through channels.

While infinite behaviour is accounted for also in IMP and HOFL (consider, e.g., the programs **rec** $x.\ x$ and **while** `true` **do skip**), unlike the sequential languages, CCS does not assign the same semantics to all the infinite behaviours (recall that if a sequential program does not terminate its semantics is equal to $\bot$ in the denotational semantics).

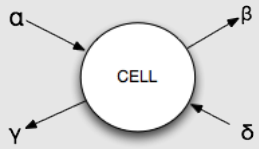
The semantics of sequential languages can be given by defining functions. In the presence of non-deterministic behaviours functions do not seem to provide the right tool to abstract the behaviour of concurrent systems. As we will see this problem is worked out by modelling the system behaviour as a *labelled transition system*, i.e. as a set of states equipped with a transition relation which keeps track of the interactions between the system and its environment. As a consequence, it makes little sense to talk about denotational semantics of CCS. In addition, recall that the denotational semantics is based on fix point theory over CPOs, while it turns out that several interesting properties of non-deterministic systems with non-trivial infinite behaviours are not inclusive (as it is the case of fairness, described in Example 5.14), thus the principle of computational induction does not apply to such properties. Moreover labelled transition systems are often equipped with a modal logic counterpart, which allows to express and prove the relevant properties of the modelled system.

Let us show how CCS works with an example.

Example 10.1 (Dynamic concurrent stack)
Let us consider the problem of modelling an extensible stack. The idea is to represent the stack as a collection of cells that communicate by sending and receiving data over some channels:

- *the* send *operation of data x over channel α is denoted by $\bar{\alpha}x$;*
- *the* receive *operation of data x over channel α is denoted by αx.*

We have one so-called process *(or* agent*) for each cell of the stack. Each process can store one or two values or send a stored value to other processes. All processes involved in the stack have basically the same structure. We represent graphically one of such processes as follow:*



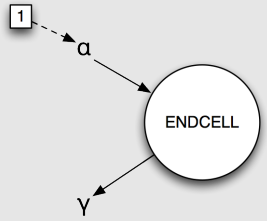
The figure shows that a cell has four channels $\alpha, \beta, \gamma, \delta$ that can be used to communicate with other cells. In general, a process can perform bidirectional operation on its channels. In this particular case, each cell will use each channel for either input or output operations:

- *α is the input channel to receive data from either the external environment or the left cell;*
- *γ is the channel used to send data to either the external environment or the left cell;*
- *β is the channel used to send data to the right cell and to manage the end of the stack;*
- *δ is the channel used to receive data from the right cell and to manage the end of the stack.*

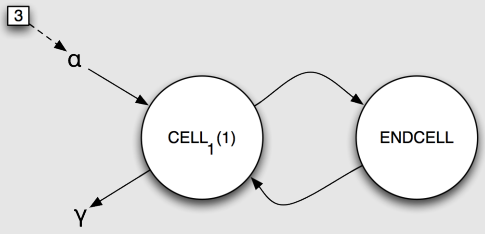
In the following we specify the possible states that a cell can have, each corresponding to some specific behaviour. Note that some states are parametric to certain values that represent, e.g., the particular values stored in that cell. The four possible states are described below:

- $CELL_0 = \delta x.\,\textbf{if}\ x = \$\ \textbf{then}\ ENDCELL\ \textbf{else}\ CELL_1(x)$
  This state represents the empty cell. The agent waits for data from the channel δ, when a value is received the agent controls if it is equal to a special termination character \$. If the received data is \$ this means that the agent is the last cell, so it switches to the ENDCELL state. Otherwise, if x is a new value, the agent stores it by moving to the state $CELL_1(x)$.
- $CELL_1(y) = \alpha x.CELL_2(x, y) + \bar{\gamma} y.CELL_0$
  This state represents an agent which contains a value y. In this case the cell can non-deterministically wait for new data on α or send the stored data on γ. In the first case, the cell must store the new value and send the old value to another agent: this task is performed by $CELL_2(x, y)$. In the second case, it is assumed that some other agent wants to extract the stored value; then the cell becomes empty by switching to the state $CELL_0$. Note that the + operator represents a non-deterministic choice performed by the agent. However a particular choice could be forced on a cell by the behaviour of the other cells.
- $CELL_2(x, y) = \bar{\beta} y.CELL_1(x)$
  In this case the cell has currently two parameters x (the newly received value) and y (the previously stored value). The agent must cooperate with its neighbors cells in order to perform a right shift of the data. In order to do that the agent communicates to the right neighbour the old stored value y and moves to state $CELL_1(x)$.
- $ENDCELL = \alpha x.(CELL_1(x) \underbrace{\frown ENDCELL}_{\text{a new bottom cell}}) + \bar{\gamma}\$.\underbrace{\textbf{nil}}_{\text{termination}}$
  This state represents the bottom of the stack. An agent in this state can perform two actions in a non-deterministic way. First it can wait for a new value (in order to perform a right shift), then store the new data and generate a new agent witch represents the new bottom element. Note that the newly created cell ENDCELL will be able to communicate with $CELL_1(x)$ only, because they will have dedicated channels. We will explain later how this can be achieved, when giving the exact definition of the operation $\frown$. Alternatively, the agent can send the special character \$ to the left cell, provided it is able to receive this character. If so, than the left cell is empty and after receiving the \$ character it becomes the new ENDCELL. Then the present agent terminates.

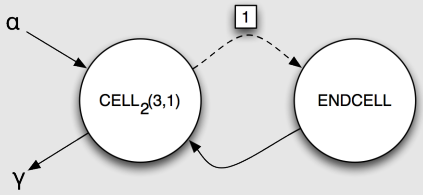
Now we will show how the stack works. Let us start from an empty stack. We have only one cell in the state ENDCELL, whose channels β and δ are made private, written $ENDCELL\backslash\beta\backslash\delta$, because on the "right" side there will be no communication. We perform a push operation in order to fill the stack with a new value. So we send the new value (1 in this case) through the channel α of the cell.



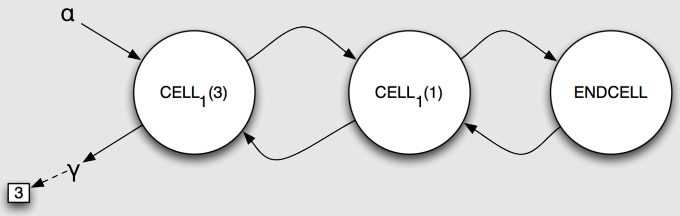
Once the cell receives the new value it generates a new bottom for the stack and changes its state to $CELL_1(1)$ *storing the new value. The result of this operation is the following:*



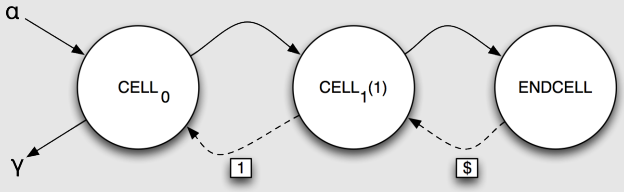
When the stack is stabilized we perform another push, this time with value 3*. In this case the first cell changes its state to* $CELL_2(3, 1)$ *in order to perform a right shift.*



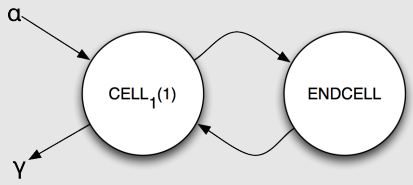
Then, when the second cell receives the value 1 *on his* α' *channel changing its state to* $CELL_1(1)$*, the first cell can stabilize itself on the state* $CELL_1(3)$*. Now we perform a pop operation, which will return the last value pushed into the stack (i.e.* 3*).*



In order to do this we read the value 3 *from the channel* γ *of the first cell. In this case the first cell changes its state to* $CELL_0$*, waiting for a value through the channel* δ*.*



When the second cell become aware of the reading performed by the first cell, it changes its state to $CELL_0$*, and reads the value sent from the third cell. Then, since the received value from ENDCELL is* \$*, it changes its state to ENDCELL. Finally, since a reading operation on* γ'' *have been performed by the second cell, the third cell reduces to* **nil**. *The situation reached after the stabilization of the system is the following:*



The above example shows that processes can synchronize in pairs, by performing dual (input/output) operations. In the following we will present a *pure* version of CCS, where we abstract away from the values communicated on channels.

10.1. Syntax of CCS

The CCS process algebra was introduced by Robin Milner in the early eighties. When presenting the syntax of CCS we will use the following conventions:

$\Delta ::= \alpha, \beta, ...$	Channels and (by coercion) input actions on channels
$\overline{\Delta} ::= \overline{\alpha}, \overline{\beta}, ...$ with $\overline{\Delta} \cap \Delta = \varnothing$	Output actions on channels
$\Lambda ::= \Delta \cup \overline{\Delta}$	Observable actions
$\tau \notin \Lambda$	Unobservable action

We extend the "bar" operation to all the elements in Λ by letting $\overline{\overline{\alpha}} = \alpha$ for all $\alpha \in \Delta$. As we have seen in the example, pairs of dual actions (e.g., α and $\overline{\alpha}$) are used to synchronize two processes. The unobservable action τ denotes a special action that is internal to some agent and that cannot be used to synchronize. Moreover we will use the following convention:

$\mu \in \Lambda \cup \{\tau\}$	generic action
$\lambda \in \Lambda$	generic channel
$\overline{\lambda} \in \Lambda$	generic dual channel

Now we are ready to present the syntax of CCS.

$$p, q ::= x \quad | \quad \mathbf{nil} \quad | \quad \mu.p \quad | \quad p\backslash\alpha \quad | \quad p[\Phi] \quad | \quad p + q \quad | \quad p \mid q \quad | \quad \mathbf{rec}\, x.p$$

x represents a process name;

nil is the empty (*inactive*) process;

$\mu.p$ is a process p *prefixed* by the action μ;

$p\backslash\alpha$ is a *restricted* process; it allows to hide the channel α from an observer external to p;

$p[\Phi]$ is a process that behaves like the process obtained from p by applying to it the permutation (a bijective substitution) Φ of its channel names. However this operation is part of the syntax and $p[\Phi]$ is syntactically different than p with the substitution performed on it. Notice that $\Phi(\tau) = \tau$;

$p + q$ is a process that can choose non-deterministically to execute either the process p or q;

$p|q$ is the process obtained as the parallel composition of p and q; the actions of p and q can be interleaved and also synchronized;

rec $x.p$ is a recursively defined process.

As usual we will consider the closed terms of this language, i.e., the processes whose process names x are all bound by recursive definitions.

10.2. Operational Semantics of CCS

The operational semantics of CCS is defined by a suitable *Labelled Transition System* (LTS) whose states are CCS (closed) processes and whose transitions are labelled by actions in $\Lambda \cup \{\tau\}$. The LTS is defined by a rule system whose formulas take the form $p_1 \xrightarrow{\mu} p_2$ meaning that the process p_1 can perform the action μ and reduce to p_2. While the LTS is the same for all CCS closed terms, starting from a CCS closed term p and using the rules we can define the LTS which represents the operational behaviour of p by considering only processes that are reachable from the state p. Although a term can be the parallel composition of many processes, its operational semantics is represented by a single global state in the LTS. Therefore concurrency and interaction between cooperating agents are not adequately represented in our CCS semantics. Now we introduce the inference rules for CCS:

$$\text{(Act)} \quad \frac{}{\mu.p \xrightarrow{\mu} p}$$

There is only one axiom in the rule system, related to the action prefix operator. It states that the process $\mu.p$ can perform the action μ and reduce to p.

$$\text{(Res)} \quad \frac{p \xrightarrow{\mu} q}{p\backslash\alpha \xrightarrow{\mu} q\backslash\alpha} \; \mu \neq \alpha, \bar{\alpha}$$

If the process is executed under a restriction, then it can perform only actions that do not involve the restricted name. Note that this restriction does not affect the communication internal to the processes, i.e., when $\mu = \tau$ the move cannot be blocked by the restriction.

$$\text{(Rel)} \quad \frac{p \xrightarrow{\mu} q}{p[\Phi] \xrightarrow{\Phi(\mu)} q[\Phi]}$$

For Φ a permutation of channel names, if p can evolve to q by performing μ, then $p[\Phi]$ can evolve to $q[\Phi]$ by performing $\Phi(\mu)$, i.e., the action μ renamed according to Φ. We remind that the unobservable action cannot be renamed, i.e., $\Phi(\tau) = \tau$ for any Φ.

$$\text{(Sum)} \quad \frac{p \xrightarrow{\mu} p'}{p + q \xrightarrow{\mu} p'} \qquad \frac{q \xrightarrow{\mu} q'}{p + q \xrightarrow{\mu} q'}$$

This pair of rules deals with non-deterministic choice: process $p + q$ can choose non-deterministically to behave like either process p or q. Moreover note that the choice can be performed only during communication, so in order to discard, e.g., process q, process p must be capable to perform an action μ. Note that axioms of the form $p + q \to p$ and $p + q \to q$ would yield a rather different semantics.

$$\text{(Com)} \quad \frac{p \xrightarrow{\mu} p'}{p|q \xrightarrow{\mu} p'|q} \qquad \frac{q \xrightarrow{\mu} q'}{p|q \xrightarrow{\mu} p|q'}$$

Also in the case of parallel composition some form of non-determinism appears. But unlike the previous case, here non-determinism is needed to simulate the parallel behaviour of the system: in the previous rule non-determinism was a characteristic of the modelled system, in this case it is a characteristic of the semantic style that allows p and q to interleave their actions in $p|q$.

$$\frac{p_1 \xrightarrow{\lambda} p_2 \quad q_1 \xrightarrow{\overline{\lambda}} q_2}{p_1|q_1 \xrightarrow{\tau} p_2|q_2}$$

There is a third rule for parallel composition, which allows processes to perform internal synchronizations. The processes p_1 and p_2 communicate by using the channel λ, which is hidden after the synchronization by using the action τ. In general, if p_1 and p_2 can perform α and $\overline{\alpha}$, respectively, then their parallel composition can perform α, $\overline{\alpha}$ or τ. When parallel composition is used in combination with the restriction operator, like in $(p_1|p_2)\backslash\alpha$, then we can force synchronization on α.

$$\text{(Rec)} \qquad \frac{p[\mathbf{rec}\ x.p/x] \xrightarrow{\mu} q}{\mathbf{rec}\ x.p \xrightarrow{\mu} q}$$

The semantics of recursion is similar to the one we have presented for HOFL: to see which moves $\mathbf{rec}\ x.p$ can perform we inspect the process $p[\mathbf{rec}\ x.p/x]$ obtained from p by replacing all free occurrences of the process name x with its full recursive definition $\mathbf{rec}\ x.p$.

We will restrict our attention to the class of *guarded agents*, namely agents in which in case of recursive terms of the form $\mathbf{rec}\ x.p$, each free occurrence of x in p occurs under an action prefix. This allows us to exclude terms like $\mathbf{rec}\ x.(x \mid p)$ which can lead (in one step) to infinitely many parallel repetitions of the same agent, making the LTS infinitely branching.

Example 10.2 (Derivation)
Let us show an example of the use of the derivation rules which we have just introduced. Take the following CCS term:

$$(((\mathbf{rec}\ x.\ \alpha.x + \beta.x) \mid (\mathbf{rec}\ x.\ \alpha.x + \gamma.x)) \mid\ \mathbf{rec}\ x.\ \bar{\alpha}.x)\backslash\alpha$$

First, let us focus on the behaviour of the (deterministic) agent $\mathbf{rec}\ x.\ \bar{\alpha}.x$.

$$\begin{aligned} \mathbf{rec}\ x.\ \bar{\alpha}.x &\xrightarrow{\bar{\alpha}} q \qquad \nwarrow_{Rec} \\ \bar{\alpha}.(\mathbf{rec}\ x.\ \bar{\alpha}.x) &\xrightarrow{\bar{\alpha}} q \qquad \nwarrow_{Act,\ q=\mathbf{rec}\ x.\ \bar{\alpha}.x} \end{aligned}$$

□

Thus:

$$\mathbf{rec}\ x.\ \bar{\alpha}.x \xrightarrow{\bar{\alpha}} \mathbf{rec}\ x.\ \bar{\alpha}.x$$

There are no other rules applicable during the above derivation; thus, the LTS associated with $\mathbf{rec}\ x.\ \bar{\alpha}.x$ *consists of a single state and one looping arrow with label* $\bar{\alpha}$. *Correspondingly, the agent is able to perform the action* $\bar{\alpha}$ *indefinitely. However, when embedded in the larger system above, then the action* $\bar{\alpha}$ *is blocked by the topmost restriction* $\_\ \backslash\alpha$. *Therefore, the only opportunity for* $\mathbf{rec}\ x.\ \bar{\alpha}.x$ *to act is by synchronizing on channel* α *with either one or the other of the two non-deterministic agents* $\mathbf{rec}\ x.\ \alpha.x+\beta.x$ *and* $\mathbf{rec}\ x.\ \alpha.x+\gamma.x$. *In fact the synchronization produces an action* τ *which cannot be blocked by* $\_\ \backslash\alpha$. *Note that each of the two non-deterministic agents is also available to interact with some external agent on another non-restricted channel, respectively* β *or* γ.

By using the rules of the operational semantics of CCS we have, e.g.:

$$\begin{array}{rl}
(((\mathbf{rec}\ x.\ \alpha.x+\beta.x) \mid (\mathbf{rec}\ x.\ \alpha.x+\gamma.x)) \mid \mathbf{rec}\ x.\ \bar{\alpha}.x) \setminus \alpha \xrightarrow{\mu} q & \nwarrow_{Res,\ q=q'\setminus\alpha} \\
((\mathbf{rec}\ x.\ \alpha.x+\beta.x) \mid (\mathbf{rec}\ x.\ \alpha.x+\gamma.x)) \mid \mathbf{rec}\ x.\ \bar{\alpha}.x \xrightarrow{\mu} q', \mu \neq \alpha, \overline{\alpha} & \nwarrow_{Com\ 3rd\ rule,\ \mu=\tau, q'=q_1 \mid q_2} \\
(\mathbf{rec}\ x.\ \alpha.x+\beta.x) \mid \mathbf{rec}\ x.\ \alpha.x+\gamma.x \xrightarrow{\lambda} q_1, \quad \mathbf{rec}\ x.\ \bar{\alpha}.x \xrightarrow{\bar{\lambda}} q_2 & \nwarrow_{Com\ 2nd\ rule,\ q_1=(\mathbf{rec}\ x.\ \alpha.x+\beta.x) \mid q_3} \\
\mathbf{rec}\ x.\ \alpha.x+\gamma.x \xrightarrow{\lambda} q_3, \quad \mathbf{rec}\ x.\ \bar{\alpha}.x \xrightarrow{\bar{\lambda}} q_2 & \nwarrow_{Rec} \\
\alpha.(\mathbf{rec}\ x.\ \alpha.x+\gamma.x)+\gamma.(\mathbf{rec}\ x.\ \alpha.x+\gamma.x) \xrightarrow{\lambda} q_3, \quad \mathbf{rec}\ x.\ \bar{\alpha}.x \xrightarrow{\bar{\lambda}} q_2 & \nwarrow_{Sum} \\
\alpha.(\mathbf{rec}\ x.\ \alpha.x+\gamma.x) \xrightarrow{\lambda} q_3, \quad \mathbf{rec}\ x.\ \bar{\alpha}.x \xrightarrow{\bar{\lambda}} q_2 & \nwarrow_{Act,\ q_3=\mathbf{rec}\ x.\ \alpha.x+\gamma.x,\ \lambda=\alpha} \\
\mathbf{rec}\ x.\ \bar{\alpha}.x \xrightarrow{\bar{\alpha}} q_2 & \nwarrow_{Rec} \\
\bar{\alpha}.(\mathbf{rec}\ x.\ \bar{\alpha}.x) \xrightarrow{\bar{\alpha}} q_2 & \nwarrow_{Act,\ q_2=\mathbf{rec}\ x.\ \bar{\alpha}.x} \\
\square &
\end{array}$$

So we have:

$$\begin{array}{l}
q_2 = \mathbf{rec}\ x.\ \bar{\alpha}.x \\
q_3 = \mathbf{rec}\ x.\ \alpha.x+\gamma.x \\
q_1 = (\mathbf{rec}\ x.\ \alpha.x+\beta.x) \mid q_3 = (\mathbf{rec}\ x.\ \alpha.x+\beta.x) \mid \mathbf{rec}\ x.\ \alpha.x+\gamma.x \\
q' = q_1 \mid q_2 = ((\mathbf{rec}\ x.\ \alpha.x+\beta.x) \mid (\mathbf{rec}\ x.\ \alpha.x+\gamma.x)) \mid \mathbf{rec}\ x.\ \bar{\alpha}.x \\
q = q'\setminus\alpha = (((\mathbf{rec}\ x.\ \alpha.x+\beta.x) \mid (\mathbf{rec}\ x.\ \alpha.x+\gamma.x)) \mid \mathbf{rec}\ x.\ \bar{\alpha}.x)\setminus\alpha \\
\mu = \tau
\end{array}$$

and thus:

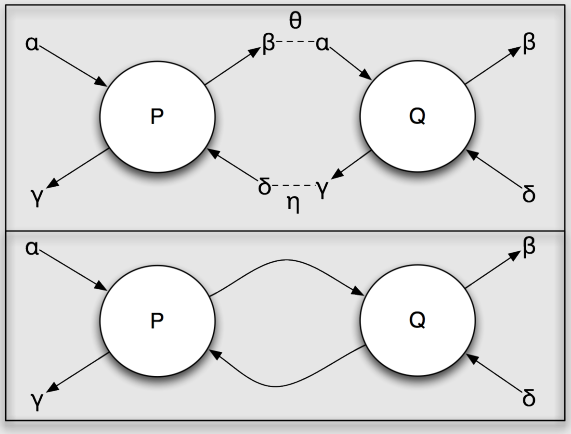
$$(((\mathbf{rec}\ x.\ \alpha.x+\beta.x) \mid (\mathbf{rec}\ x.\ \alpha.x+\gamma.x)) \mid \mathbf{rec}\ x.\ \bar{\alpha}.x)\setminus\alpha \xrightarrow{\tau} (((\mathbf{rec}\ x.\ \alpha.x+\beta.x) \mid (\mathbf{rec}\ x.\ \alpha.x+\gamma.x)) \mid \mathbf{rec}\ x.\ \bar{\alpha}.x)\setminus\alpha$$

Note that during the derivation we had to choose several times between different rules which could be applied; while in general it may happen that wrong choices can lead to dead ends, our choices have been made so to complete the derivation satisfactorily, avoiding any backtracking.

Example 10.3 (Dynamic stack continuation)
Let us conclude the dynamic stack example by formalizing in CCS the concatenation operator:

$$P \frown Q = (P[^{\vartheta}/_{\beta},^{\eta}/_{\delta}] \mid Q[^{\vartheta}/_{\alpha},^{\eta}/_{\gamma}])\setminus\vartheta\setminus\eta$$

where ϑ and η are two new hidden channels shared by the processes. Note that the locality of these names allows to avoid conflicts between channel's names.



10.3. Abstract Semantics of CCS

As we saw each CCS agent can be represented by a LTS, i.e., by a labelled graph. It is easy to see that such operational semantics is much more concrete and detailed than the semantics studied for IMP and HOFL. For example, since the states of the LTS are named by agents it is evident that two syntactically different processes like $p|q$ and $q|p$ are associated with different graphs, even if intuitively one would expect that both exhibit the same behaviour. Thus it is important to find a good notion of equivalence, able to provide a more abstract semantics for CCS. As it happens for the denotational semantics of IMP and HOFL, an abstract semantics defined up to equivalence should abstract from the way agents execute, focusing on their external visible behaviours.

In this section we first show that neither graph isomorphism nor trace semantics are fully satisfactorily abstract semantics to capture the features of communicating systems represented in CCS. Next, we introduce a more satisfactory semantics of CCS by defining a relation, called *bisimilarity*, that captures the ability of processes to simulate each other. Finally, we discuss some positive and negative aspects of bisimilarity and present some possible alternatives.

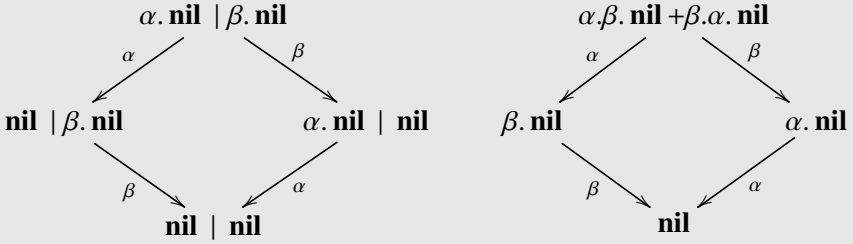
10.3.1. Graph Isomorphism

It is quite obvious to think that two agents must be considered as equivalent if their (LTS) graphs are isomorphic. Recall that two graphs are said to be isomorphic if there exists a bijection f which preserves the structure of the graphs.

Example 10.4 (Isomorphic agents)
Let us consider the following agents:

$$\alpha.\,\mathbf{nil} \mid \beta.\,\mathbf{nil} \qquad \alpha.\beta.\,\mathbf{nil} + \beta.\alpha.\,\mathbf{nil}$$

Their reachable parts of the LTS are as follows:



The two graphs are isomorphic, thus the two agents should be considered as equivalent. This result

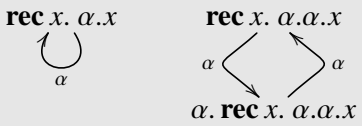
is surprising, since they have a rather different structure. In fact, the example shows that concurrency can be reduced to non-determinism by graph isomorphism. This is due to the interleaving of the actions performed by processes that are composed in parallel, which is a characteristic of the semantics which we have presented.

This approach is very simple and natural but still leads to semantics that is too concrete, i.e., graph isomorphism still distinguishes too much. We show this fact in the following example.

Example 10.5
Let us consider the agents:

$$\mathbf{rec}\ x.\ \alpha.x \qquad \mathbf{rec}\ x.\ \alpha.\alpha.x$$

Their reachable parts of the LTS are as follows:



The two graphs are not isomorphic, but it is hardly possible to distinguish between the two agents according to their behaviour: they both are able only to execute any sequence of α.

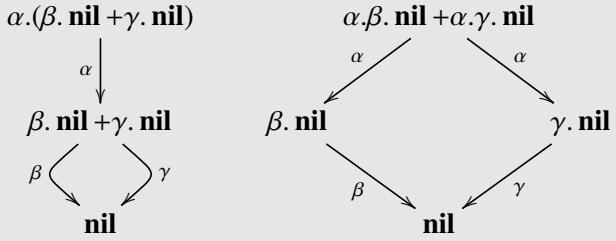
10.3.2. Trace Equivalence

A second approach, trace equivalence, observes the set of traces of an agent, namely the set of sequences of actions labelling any path in its graph. Trace equivalence is strictly coarser than equivalence based on graph isomorphism, since isomorphic graphs have the same traces. Conversely, Example 10.5 shows two agents which are trace equivalent but whose graphs are not isomorphic. In fact, trace equivalence is too coarse: the following example shows that trace equivalence is not able to capture the choice points within agent behaviour.

Example 10.6
Let us consider the following agents:

$$p = \alpha.(\beta.\ \mathbf{nil} + \gamma.\ \mathbf{nil}) \qquad q = \alpha.\beta.\ \mathbf{nil} + \alpha.\gamma.\ \mathbf{nil}$$

Their reachable parts of the LTS are as follows:



These two graphs are trace equivalent: the trace sets are both $\{\alpha\beta, \alpha\gamma\}$. However the agents do not behave in the same way if we regard the choices they make. In the second agent the choice between β and γ is made during the first step, by selecting one of the two possible α. In the first agent, on the contrary, the same choice is made in a second time, after the execution of the unique α action.

The difference is evident if we consider, e.g., that an agent $\bar{\alpha}.\bar{\beta}.\mathbf{nil}$ may be running in parallel, with actions α, β and γ restricted on top: the agent p is always able to carry out the complete interaction with $\bar{\alpha}.\bar{\beta}.\mathbf{nil}$, because after the synchronization on α is ready to synchronize on β; vice versa, the agent q is only able to carry out the complete interaction with $\bar{\alpha}.\bar{\beta}.\mathbf{nil}$ if the left choice is performed at the time of the first interaction on α, as otherwise $\gamma.\mathbf{nil}$ and $\bar{\beta}.\mathbf{nil}$ cannot interact. Formally, if we consider the context $C(\_) = (\_ \mid \overline{\alpha}.\overline{\beta}.nil) \setminus \alpha \setminus \beta \setminus \gamma$ we have that $C(q)$ can deadlock, while $C(p)$ cannot. Figure out how embarrassing could be the difference if α would mean for a computer to ask if a file should be deleted, and β, γ were the user yes/no answer: p would behave as expected, while q could decide to delete the file in the first place, and then deadlock if the the user decides otherwise. See also Example 10.20.

Given all the above, we can argue that neither graph isomorphism nor trace equivalence are good candidates for our behavioural equivalence relation. Still, it is obvious that: 1) isomorphic agents must be retained as equivalent; 2) equivalent agents must be trace equivalent. Thus, our candidate equivalence relation must be situated in between graph isomorphism and trace equivalence.

10.3.3. Bisimilarity

In this section we introduce a class of relations between agents called *bisimulations* and we construct a behavioural equivalence relation between agents called *bisimilarity* as the largest bisimulation. This equivalence relation is shown to be the one we were looking for, namely the one that identifies only those agents which intuitively have the same behaviour.

Let us start with an example which illustrates how bisimilarity works.

Example 10.7 (Game Theory)
In this example we use game theory in order to show that the agents of the example 10.6 are not behaviourally equivalent. In our example game theory is used in order to prove that a system verifies or not a property. We can imagine two player called Alice and Bob, the goal of Alice is to prove that the system has not the property. Bob, on the contrary, wants to show that the system satisfies the property. The game starts and at each turn each player can make a move in order to reach his/her goal. At the end of the game if Alice wins this means that the system does not satisfy the property. If the winner is Bob, instead, the system satisfies the property.

We apply this pattern to the states of LTS which describe CCS agents. Let us take two states p and q of a LTS. Alice would like to show that p is not behavioural equivalent to q, Bob on the other hand would like to show that p and q have the same behaviour.

Alice starts the game. At each turn of the game Alice executes (if possible) a transition of the transition system of either p or q and Bob must execute a transition with the same label but of the other agent. If Alice cannot move on both p and q, then Alice has lost, since this means that p and q are both deadlocked, and thus obviously equivalent. Alice wins if she can make a move that Bob cannot imitate; or if she has a move which, no matter which is the answer by Bob, will lead to a situation where she can make a move that Bob cannot imitate; or ... and so on for any number of moves. Bob wins if Alice has no such a (finite) strategy. Note that the game does not necessarily terminate: also in this case Bob wins, that is p and q are equivalent.

In the example 10.6, let us take $p = \alpha.(\beta.\mathbf{nil} + \gamma.\mathbf{nil})$ and $q = \alpha.\beta.\mathbf{nil} + \alpha.\gamma.\mathbf{nil}$. Alice starts by choosing p and by executing the only transition labelled α. Then, Bob can choose one of the two transitions labelled α leaving from q. Suppose that Bob chooses the left α transition (but the case where Bob chooses the right transition leads to the same result of the game). So the reached states are $\beta.\mathbf{nil} + \gamma.\mathbf{nil}$ and $\beta.\mathbf{nil}$. In the second turn Alice chooses the transition labelled γ from $\beta.\mathbf{nil} + \gamma.\mathbf{nil}$, and Bob can not simulate this execution. Since Alice has a winning, two-moves strategy, the two agents are not equivalent.

Now we define the same relation in a more formal way, as originally introduced by Robin Milner. It is important to notice that the definition applies to a generic labelled transition systems, namely a set of states P equipped with a ternary relation $\longrightarrow \subseteq P \times L \times P$, where L is a generic set of actions. As we have seen, a unique LTS is associated to CCS, where CCS agents are states and a triple (p, α, q) belongs to $\longrightarrow$ iff $p \xrightarrow{\alpha} q$ is a theorem of the operational semantics. Notice that agents with isomorphic graphs are automatically equivalent.

Definition 10.8 (Strong Bisimulation)
Let R be a binary relation on the set of states of an LTS then it is a strong bisimulation *if*

$$\forall s_1\ R\ s_2 \Rightarrow \begin{array}{ll} \text{if } s_1 \xrightarrow{\alpha} s_1' & \text{then there exists a transition } s_2 \xrightarrow{\alpha} s_2' \text{ such that } s_1'\ R\ s_2' \\ \text{if } s_2 \xrightarrow{\alpha} s_2' & \text{then there exists a transition } s_1 \xrightarrow{\alpha} s_1' \text{ such that } s_1'\ R\ s_2' \end{array}$$

For example it is easy to check that the identity relation $\{(p, p) \mid p \text{ is a CCS process }\}$ is a strong bisimulation, that graph isomorphism is a strong bisimulation and that the union $R_1 \cup R_2$ of two strong bisimulation relations R_1 and R_2 is also a strong bisimulation relation. Moreover, given the composition of relations defined by

$$R_1 \circ R_2 \stackrel{\text{def}}{=} \{(p, p') \mid \exists p''.p\ R_1\ p'' \wedge p''\ R_2\ p'\}$$

it can be shown that $R_1 \circ R_2$ is a strong bisimulation when R_1 and R_2 are such.

Definition 10.9 (Strong bisimilarity $\simeq$)
Let s and s′ be two states of a LTS, then they are said to be bisimilar *and write $s \simeq s'$ if and only if there exists a strong bisimulation R such that $s\ R\ s'$.*

The relation $\simeq$ is called strong bisimilarity *and is defined as follows:*

$$\simeq \stackrel{\text{def}}{=} \bigcup_{R \text{ is a strong bisimulation}} R$$

Strong bisimilarity $\simeq$ is an equivalence relation on CCS processes. Below we recall the definition of equivalence relation.

Definition 10.10 (Equivalence Relation)
Let $\equiv$ be a binary relation on a set X, then we say that it is an equivalence relation *if it has the following properties:*

- $\forall x, y \in X.\ x \equiv x$ *(Reflexivity)*
- $\forall x, y, z \in X.\ x \equiv y \wedge y \equiv z \Rightarrow x \equiv z$ *(Transitivity)*
- $\forall x, y \in X.\ x \equiv y \Rightarrow y \equiv x$ *(Symmetry)*

Definition 10.11 (Equivalence Class and Quotient Set)
Given an equivalence relation $\equiv$ on X and an element x of X we call equivalence class *of x the subset $[x]$ of X defined as follows:*

$$[x] = \{y \in X \mid x \equiv y\}$$

The set $X_{/\equiv}$ containing all the equivalence classes generated by a relation $\equiv$ on the set X is called quotient set.

We omit the proof of the following theorem that is based on the above mentioned properties of strong bisimulations and on the fact that bisimilarity is a strong bisimulation.

Theorem 10.12
The bisimilarity relation $\simeq$ is an equivalence relation between CCS agents.

Now we will use the fixpoint theory, which we have introduced in the previous chapters, in order to define bisimilarity in a more effective way. Using fixpoint theory we will construct, by successive approximations, the coarsest (maximal) bisimulation between the states of a LTS, which is actually an equivalence relation.

As usual, we define the $CPO_\perp$ on which the approximation function works. The $CPO_\perp$ is defined on the set $\mathscr{P}(P \times P)$, namely the power set of the pairs of states of the LTS. As we saw in the previous chapters the pair $(\mathscr{P}(P \times P), \subseteq)$ (all the subsets of a given set, ordered by inclusion) is a $CPO_\perp$, however it is not the one which we will use. As we said we would like to start from the roughest relation, which considers all the states equivalent and, by using the fixpoint operator, to reach the relation which will identify only bisimilar agents. So we need a $CPO_\perp$ in which a set with more elements is considered smaller than one with few elements. Thus we define the order relation $R \sqsubseteq R' \Leftrightarrow R' \subseteq R$ between subsets of $\mathscr{P}(P \times P)$. The resulting $CPO_\perp$ $(\mathscr{P}(P \times P), \sqsubseteq)$ is represented in figure 10.1 .

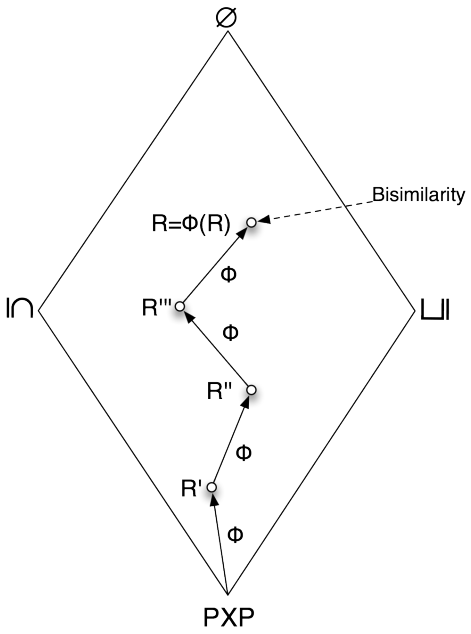


Figure 10.1.: $CPO_\perp$ $(\mathscr{P}(P \times P), \sqsubseteq)$

Now we define the function $\Phi : \mathscr{P}(P \times P) \to \mathscr{P}(P \times P)$ on relations on P.

$$p\ \Phi(R)\ q \stackrel{\text{def}}{=} \begin{cases} p \xrightarrow{\mu} p' \Longrightarrow \exists q'.\ q \xrightarrow{\mu} q' & \text{and} \quad p'Rq' \\ q \xrightarrow{\mu} q' \Longrightarrow \exists p'.\ p \xrightarrow{\mu} p' & \text{and} \quad p'Rq' \end{cases}$$

Definition 10.13 (Bisumulation as fixpoint)
Let R be a relation in $\mathscr{P}(P \times P)$ then it is said to be a bisimulation *iff:*

$$\Phi(R) \sqsubseteq R \quad (i.e.,\ R \subseteq \Phi(R))$$

Theorem 10.14 (Bisimilarity as fixpoint)
The function Φ is monotone and continue, i.e.:

$$\begin{array}{rcl} R_1 \sqsubseteq R_2 & \Rightarrow & \Phi(R_1) \sqsubseteq \Phi(R_2) \\ \Phi(\bigsqcup_{n\in\omega} R_n) & = & \bigsqcup_{n\in\omega} \Phi(R_n) \end{array}$$

Moreover, the least fixed point of Φ is the bisimilarity, namely it holds:

$$\simeq \stackrel{\text{def}}{=} \bigsqcap_{R=\Phi(R)} R = \bigsqcup_{n\in\omega} \Phi^n(P \times P)$$

We do not prove the above theorem. Note that monotonicity is obvious, since a larger R will make the conditions on $\Phi(R)$ weaker. Continuity of Φ is granted only if the LTS is finitely branching, namely if every state has a finite number of outgoing transitions. If Φ is not continuous, we still have the existence of a minimal fixpoint, but, it will not be always reachable by the ω chain of approximations.

Example 10.15 (Infinitely branching process)
Let us consider the agent $p = \mathbf{rec}\, x.\, (x \mid \alpha.\mathbf{nil})$. The agent p is not guarded, because the occurrence of x in the body of the recursive process is not prefixed by an action. By using the rules of the operational semantics of CCS we have, e.g.:

$$\begin{array}{rl} \mathbf{rec}\, x.\, (x \mid \alpha.\mathbf{nil}) \xrightarrow{\mu} q & \nwarrow_{Rec} \\ (\mathbf{rec}\, x.\, (x \mid \alpha.\mathbf{nil})) \mid \alpha.\mathbf{nil} \xrightarrow{\mu} q & \nwarrow_{Com\ 1st\ rule,\ q=q_1 \mid \alpha.\mathbf{nil}} \\ \mathbf{rec}\, x.\, (x \mid \alpha.\mathbf{nil}) \xrightarrow{\mu} q_1 & \nwarrow_{Rec} \\ (\mathbf{rec}\, x.\, (x \mid \alpha.\mathbf{nil})) \mid \alpha.\mathbf{nil} \xrightarrow{\mu} q_1 & \nwarrow_{Com\ 1st\ rule,\ q_1=q_2 \mid \alpha.\mathbf{nil}} \\ \mathbf{rec}\, x.\, (x \mid \alpha.\mathbf{nil}) \xrightarrow{\mu} q_2 & \nwarrow_{Rec} \\ \ldots & \\ \mathbf{rec}\, x.\, (x \mid \alpha.\mathbf{nil}) \xrightarrow{\mu} q_n & \nwarrow_{Rec} \\ (\mathbf{rec}\, x.\, (x \mid \alpha.\mathbf{nil})) \mid \alpha.\mathbf{nil} \xrightarrow{\mu} q_n & \nwarrow_{Com\ 2nd\ rule,\ q_n=(\mathbf{rec}\, x.\, (x \mid \alpha.\mathbf{nil})) \mid q'} \\ \alpha.\mathbf{nil} \xrightarrow{\mu} q' & \nwarrow_{Act,\ \mu=\alpha, q'=\mathbf{nil}} \\ \Box & \end{array}$$

It is then evident that for any $n \in \omega$ we have:

$$\mathbf{rec}\, x.\, (x \mid \alpha.\mathbf{nil}) \xrightarrow{\alpha} (\mathbf{rec}\, x.\, (x \mid \alpha.\mathbf{nil})) \mid \mathbf{nil} \mid \underbrace{\alpha.\mathbf{nil} \mid \cdots \mid \alpha.\mathbf{nil}}_{n}$$

The following lemma ensures that if we consider only guarded terms then the LTS is finitely branching.

Lemma 10.16
Let p be a guarded CCS term then $\{q \mid p \xrightarrow{\mu} q\}$ is a finite set.

In order to apply the fixpoint theorem to calculate the bisimilarity, we consider only states which are reachable from the states we want to compare for bisimilarity. If the number of reachable states is finite, i.e.

if the system is a finite state automata, the calculation is effective, but possibly quite complex if the number of states is large.

Example 10.17 (Bisimilarity as fixpoint)
Let us consider the example 10.6 which we have already solved with game theory techniques. Now we show the fixpoint approach to the same system. Let us restrict the attention to the set of reachable states and represent the relations by showing the equivalence classes which they induce (over reachable processes) . We start with the coarsest relation (just one equivalence class):

$$R_0 = \{\ \{\alpha.(\beta.\,\mathbf{nil} + \gamma.\,\mathbf{nil})\ ,\ \alpha.\beta.\,\mathbf{nil} + \alpha.\gamma.\,\mathbf{nil}\ ,\ \beta.\,\mathbf{nil} + \gamma.\,\mathbf{nil}\ ,\ \beta.\,\mathbf{nil}\ ,\ \gamma.\,\mathbf{nil}\ ,\ \mathbf{nil}\}\ \}$$

By applying Φ:

$$R_1 = \Phi(R_0) = \{\ \{\alpha.(\beta.\,\mathbf{nil} + \gamma.\,\mathbf{nil}), \alpha.\beta.\,\mathbf{nil} + \alpha.\gamma.\,\mathbf{nil}\}\ ,\ \{\beta.\,\mathbf{nil} + \gamma.\,\mathbf{nil}\}\ ,\ \{\beta.\,\mathbf{nil}\}\ ,\ \{\gamma.\,\mathbf{nil}\}\ ,\ \{\mathbf{nil}\}\ \}$$

$$R_2 = \Phi(R_1) = \{\ \{\alpha.(\beta.\,\mathbf{nil} + \gamma.\,\mathbf{nil})\}\ ,\ \{\alpha.\beta.\,\mathbf{nil} + \alpha.\gamma.\,\mathbf{nil}\}\ ,\ \{\beta.\,\mathbf{nil} + \gamma.\,\mathbf{nil}\}\ ,\ \{\beta.\,\mathbf{nil}\}\ ,\ \{\gamma.\,\mathbf{nil}\}\ ,\ \{\mathbf{nil}\}\ \}$$

Note that R_2 is a fixpoint, hence it is the coarsest bisimulation.

10.4. Compositionality

In this section we focus our attention on the *compositionality* aspect of the abstract semantics which we have just introduced. For an abstract semantics to be practically relevant it is important that any process used in a system can be replaced with an equivalent process without changing the semantics of the system. Since we have not used structural induction in defining the abstract semantics of CCS, no one ensures any kind of compositionality w.r.t. the possible way of constructing larger systems, i.e., w.r.t. the operators of CCS.

Definition 10.18 (Context)
A context *is a term with a gap which can be filled by inserting any other term of our language. We write $C[\ ]$ to indicate a* context.

Definition 10.19 (Congruence)
A relation $\sim_{\mathscr{C}}$ is said to be a congruence *(with respect to a class of contexts) if:*

$$\forall C[\ ].\ p\ \sim_{\mathscr{C}}\ q \quad \Rightarrow \quad C[p]\ \sim_{\mathscr{C}}\ C[q]$$

In order to guarantee the compositionality of CCS we must show that the bisimilarity is a congruence relation.

Let us now see an example of a relation which is not a congruence.

Example 10.20 (Trace equivalence)
Let us consider the trace equivalence relation, which we have defined in Section 10.3.2. Take the following context:

$$C[\_] = (\ \_\ \mid \bar{\alpha}.\bar{\beta}.\,\mathbf{nil})\backslash\alpha\backslash\beta\backslash\gamma$$

Now we can fill the gaps with the following terms:

$$C[p] = (\alpha.(\beta.\,\mathbf{nil} + \gamma.\,\mathbf{nil}) \mid \bar{\alpha}.\bar{\beta}.\,\mathbf{nil})\backslash\alpha\backslash\beta\backslash\gamma$$

$$C[q] = ((\alpha.\beta.\,\mathbf{nil} + \alpha.\gamma.\,\mathbf{nil}) \mid \bar{\alpha}.\bar{\beta}.\,\mathbf{nil})\backslash\alpha\backslash\beta\backslash\gamma$$

Obviously $C[p]$ and $C[q]$ generate the same set of traces, however one of the processes can "deadlock" before the interaction on β takes place, but not the other. The difference can be formalized if we consider the so-called completed trace semantics.

A completed trace *of a process p is a sequence of actions $\mu_1 \cdots \mu_k$ (for $k \geq 0$) such that there exists a sequence of transitions*

$$p = p_0 \xrightarrow{\mu_1} p_1 \xrightarrow{\mu_2} \cdots \xrightarrow{\mu_{k-1}} p_{k-1} \xrightarrow{\mu_k} p_k \not\rightarrow$$

for some $p_1, ..., p_k$. The completed traces of a process characterize the sequences of actions that can lead the system to a deadlocked configuration, where no further action is possible.

The completed trace semantics of p is the same as that of q, namely $\{\ \alpha\beta\ ,\ \alpha\gamma\ \}$. However, the completed traces of $C[p]$ and $C[q]$ are $\{\ \tau\tau\ \}$ and $\{\ \tau\tau\ ,\ \tau\ \}$, respectively. We can thus conclude that the completed trace semantics is not a congruence.

10.4.1. Bisimilarity is Preserved by Parallel Composition

In order to show that bisimilarity is a congruence we should prove that the property holds for all the operators of CCS, since this implies that the property holds for all contexts. However we show the proof only for parallel composition, which is a quite interesting case to consider. The other cases follow by similar arguments.

Formally, we need to prove that:

$$p_1 \simeq p_2 \wedge q_1 \simeq q_2 \quad \stackrel{?}{\Longrightarrow} \quad p_1 \mid q_1 \simeq p_2 \mid q_2$$

As usual we assume the premises and we would like to prove:

$$\exists R.\ (p_1 \mid q_1)\ R\ (p_2 \mid q_2)\ \wedge\ R \subseteq \Phi(R)$$

Since $p_1 \simeq p_2$ and $q_1 \simeq q_2$ we have:

$$\begin{array}{ll} p_1\ R_1\ p_2 & \text{for some bisimulation } R_1 \\ q_1\ R_2\ q_2 & \text{for some bisimulation } R_2 \end{array}$$

Now we define a bisimulation that satisfies the requested property:

$$R \stackrel{\text{def}}{=} \{(\hat{p}_1 \mid \hat{q}_1\ ,\ \hat{p}_2 \mid \hat{q}_2) \mid \hat{p}_1\ R_1\ \hat{p}_2\ \wedge\ \hat{q}_1\ R_2\ \hat{q}_2\}$$

By definition it holds $p_1 \mid q_1\ R\ p_2 \mid q_2$.
Now we show that R is a bisimulation ($R \subseteq \Phi(R)$):

$$P(p_1|q_1 \xrightarrow{\mu} p_1'|q_1') \stackrel{\text{def}}{=} \forall p_2, q_2.\ (p_1|q_1\ R\ p_2|q_2 \Longrightarrow \exists p_2', q_2'.\ p_2|q_2 \xrightarrow{\mu} p_2'|q_2'\ \wedge\ p_1'|q_1'\ R\ p_2'|q_2')$$

We proceed by rule induction. There are three possible rules for parallel composition (Com). We start by considering the rule:

$$\frac{p \xrightarrow{\mu} p'}{p|q \xrightarrow{\mu} p'|q}$$

The property for this rule is the following:

$$P(p|q \xrightarrow{\mu} p'|q) \stackrel{\text{def}}{=} \forall p_2, q_2.\ (p|q\ R\ p_2|q_2 \Longrightarrow \exists p_2', q_2'.\ p_2|q_2 \xrightarrow{\mu} p_2'|q_2'\ \wedge\ p'|q\ R\ p_2'|q_2')$$

We assume that $p \xrightarrow{\mu} p'$ and that, by definition of R, $p\ R_1\ p_2$ and $q\ R_2\ q_2$. Then we have:

$$\exists p_2'.p_2 \xrightarrow{\mu} p_2'\ \wedge\ p'\ R_1\ p_2'$$

By applying the first (Com) rule:

$$p_2|q_2 \xrightarrow{\mu} p_2'|q_2$$

By definition of R we conclude:

$$p'|q \; R \; p_2'|q_2$$

The proof for the second rule

$$\frac{q \xrightarrow{\mu} q'}{p|q \xrightarrow{\mu} p|q'}$$

is analogous.

Finally, we consider the third (Com) rule:

$$\frac{p \xrightarrow{\lambda} p' \quad q \xrightarrow{\bar{\lambda}} q'}{p|q \xrightarrow{\tau} p'|q'}$$

The property for this rule is the following:

$$P(p|q \xrightarrow{\tau} p'|q') \stackrel{\text{def}}{=} \forall p_2, q_2.\, (p|q \; R \; p_2|q_2 \Longrightarrow \exists p_2', q_2'.\, p_2|q_2 \xrightarrow{\tau} p_2'|q_2' \wedge p'|q' \; R \; p_2'|q_2')$$

Assuming the premise and by definition of R we have:

$$p \xrightarrow{\lambda} p' \qquad q \xrightarrow{\bar{\lambda}} q' \qquad p \; R_1 \; p_2 \qquad q \; R_2 \; q_2$$

Therefore:

$$\begin{array}{lllll} p \xrightarrow{\lambda} p' & \wedge & p \; R_1 \; p_2 & \Rightarrow & \exists p_2'.\, p_2 \xrightarrow{\lambda} p_2' \quad \wedge \quad p' \; R_1 \; p_2' \\ q \xrightarrow{\bar{\lambda}} q' & \wedge & q \; R_2 \; q_2 & \Rightarrow & \exists q_2'.\, q_2 \xrightarrow{\bar{\lambda}} q_2' \quad \wedge \quad q' \; R_2 \; q_2' \end{array}$$

By applying the third (Com) rule we obtain:

$$p_2|q_2 \xrightarrow{\tau} p_2'|q_2'$$

We conclude by definition of R:

$$p'|q' \; R \; p_2'|q_2'$$

10.5. Hennessy - Milner Logic

In this section we present a *modal logic* introduced by Matthew Hennessy and Robin Milner. Modal logic allows to express concepts as "there exists a next state such that", or "for all next states", some property holds. Typically, model checkable properties are stated as formulas in some modal logic. In particular, Hennessy-Milner modal logic is relevant for its simplicity and for its close connection with bisimilarity. As we will see, in fact, two bisimilar agents verify the same set of modal logic formulas. This fact shows that bisimilarity is at the right level of abstraction.

First of all we introduce the syntax of the *Hennessy-Milner logic* (HM-logic):

$$F ::= true \quad | \quad \neg F \quad | \quad \wedge_{i \in I} F_i \quad | \quad \Diamond_\mu F$$

We write $\mathscr{L}$ for the set of the HM-logic formulas.

The formulas of HM-logic express properties of LTS states, namely in our case of CCS agents. The meanings of the logic operators are the following:

- *true*: is the formula satisfied by every agent. Notice that true can be considered a shorthand for an indexed conjunction $\wedge_{i \in I}$ where the set I of indexes is empty.
- $\neg F$: is the classic logic negation.

- $\wedge_{\forall i \in I} F_i$: is equivalent to the classic "and" operator applied to the set of formulas $\{F_i\}_{i \in I}$.
- $\Diamond_\mu F$: it is a *modal operator*, an agent p satisfies this formula if there exists a transition from p to q labelled with μ and the formula F holds in q.

As usual in logic satisfaction is defined as a relation $\models$ between formulas and their models, which in our case are states of a LTS.

Definition 10.21 (Satisfaction relation)
The satisfaction relation $\models \subseteq P \times \mathscr{L}$ *is defined as follows:*

$$\begin{array}{lll} p \models true & & \\ p \models \neg F & \textit{iff} & \textit{not } p \models F \\ p \models \wedge_{i \in I} F_i & \textit{iff} & p \models F_i \qquad \forall i \in I \\ p \models \Diamond_\mu F & \textit{iff} & \exists p'.\ p \xrightarrow{\mu} p' \wedge p' \models F \end{array}$$

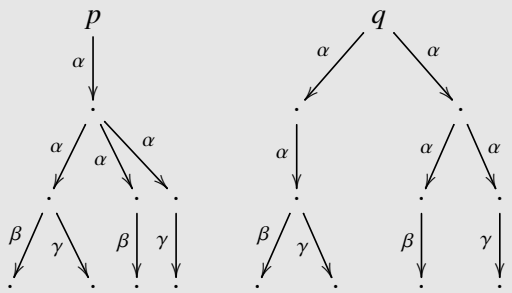
Starting from the basic operators we have just introduced we can extend our language with derived operators of common use:

- $false \stackrel{\text{def}}{=} \neg true$
- $\vee_{i \in I} F_i \stackrel{\text{def}}{=} \neg \wedge_{i \in I} \neg F_i$
- $\Box_\mu F \stackrel{\text{def}}{=} \neg \Diamond_\mu \neg F$

It is worth to comment on the meaning of the last operator: the formula $\Box_\mu F$ is valid in a state p if each transition starting in p and labelled μ reaches only states which satisfy F.

$$p \models \Box_\mu F \quad \text{iff} \quad \forall p'.\ p \xrightarrow{\mu} p' \Rightarrow p' \models F$$

Example 10.22 (non-equivalent agents)
Let us consider two CCS agents p and q associated with the following graphs:



We would like to show a formula F which is satisfied by one of the two agents and not by the other. For example we can take:

$$F = \Diamond_\alpha \Box_\alpha (\Diamond_\beta true \wedge \Diamond_\gamma true)$$

we have:

$$q \models F \qquad p \not\models F$$

In fact in q we can choose the left α-transition and we reach a state that satisfies $\Box_\alpha(\Diamond_\beta true \wedge \Diamond_\gamma true)$ (i.e., the (only) state reachable by an α-transition can perform both γ and β). On the contrary, the agent p does not satisfy the formula F because after the unique α-transition it is possible to take α-transitions that lead to states where either β or γ is enabled, but not both.

The HM-logic induces an obvious equivalence on CCS processes: two agents are logically equivalent if they satisfy the same set of formulas. Now we present two theorems which allow us to connect bisimilarity and modal logic. As we said this connection is very important both from theoretical and practical point of view. We start by introducing a measure over formulas to estimate the maximal number of consecutive steps that must be taken into account to check the validity of the formulas.

Definition 10.23 (Depth of a formula)
We define the depth *of a formula as follows:*

$$\begin{aligned} D(true) &= 0 \\ D(\neg F) &= D(F) \\ D(\wedge_{i\in I} F_i) &= \max(D(F_i) \mid i \in I) \\ D(\Diamond_\mu F) &= D(F) + 1 \end{aligned}$$

We will denote the set of logic formulas of depth k with $\mathscr{L}_k = \{F \mid D(F) = k\}$.

The first theorem ensures that if two agents are not distinguished by the k^{th} iteration of the fixpoint calculation of bisimilarity, then no formula of depth k can distinguish between the two agents, and viceversa.

Theorem 10.24
Let $\sim_k$ *be defined as follows:*

$$p \sim_k q \Leftrightarrow p\ \Phi^k(P \times P)\ q$$

and let p and q be two CCS agents. Then, we have:

$$p \sim_k q \quad \textit{iff} \quad \forall F \in \mathscr{L}_k.\ (p \models F) \Leftrightarrow (q \models F)$$

The second theorem generalizes the above correspondence by setting up a connection between formulas of any depth and bisimilarity. The proof is by induction on the depth of formulas.

Theorem 10.25
Let p and q two CCS agents, then we have:

$$p \simeq q \quad \textit{iff} \quad \forall F.(p \models F) \Leftrightarrow (q \models F)$$

It is worth reading this result both in the positive sense, namely bisimilar agents satisfy the same set of HM formulas; and in the negative sense, namely if two agents are not bisimilar, then there exists a formula which distinguishes between them. From a theoretical point of view these theorems show that bisimilarity distinguishes all and only those agents which are really different because they enjoy different properties. These results witness that the relation $\simeq$ is a good choice from the logical point of view.

10.6. Axioms for Strong Bisimilarity

Finally, we show that strong bisimilarity can be finitely axiomatized. First we present a theorem which allows to derive for every non recursive CCS agent a suitable normal form.

Theorem 10.26
Let p be a (non-recursive) CCS agent, then there exists a CCS agent strongly bisimilar to p built only with prefix, sum and nil.

Proof. We proceed by structural recursion. First define two binary operators $\rfloor$ and $\|$, where $\rfloor q$ means that q does not perform any action, and $p_1 \| p_2$ means that p_1 and p_2 must perform a synchronization. This corresponds to say that the operational semantics rules for $p \rfloor q$ and $p \| q$ are:

$$\frac{p \xrightarrow{\mu} p'}{p \rfloor q \xrightarrow{\mu} p' | q} \qquad \frac{p \xrightarrow{\lambda} p' \quad q \xrightarrow{\bar{\lambda}} q'}{p \| q \xrightarrow{\tau} p' | q'}$$

We show how to decompose the parallel operator, then we show the other cases:

$$p_1 \mid p_2 \simeq p_1 \rfloor p_2 \; + \; p_2 \rfloor p_1 \; + \; p_1 \| p_2$$

Moreover we have the following equalities:

$$\begin{aligned}
\mu.p \rfloor q &\simeq \mu.(p \mid q) \\
(p_1 + p_2) \rfloor q &\simeq p_1 \rfloor q \; + \; p_2 \rfloor q \\
\\
\mu_1.p_1 \| \mu_2.p_2 &\simeq \mathbf{nil} \;\text{ if }\; \mu_1 \neq \bar{\mu}_2 \\
\lambda.p_1 \| \overline{\lambda}.p_2 &\simeq \tau.(p_1 \mid p_2) \\
(p_1 + p_2) \| (q_1 + q_2) &\simeq p_1 \| q_1 \; + \; p_1 \| q_2 \; + \; p_2 \| q_1 \; + \; p_2 \| q_2 \\
\\
(\mu.p) \setminus \alpha &\simeq \mu.(p \setminus \alpha) \;\text{ if }\; \mu \neq \alpha, \overline{\alpha} \\
(\mu.p) \setminus \alpha &\simeq \mathbf{nil} \;\text{ if }\; \mu \in \{\alpha, \overline{\alpha}\} \\
(p_1 + p_2) \setminus \alpha &\simeq p_1 \setminus \alpha \; + \; p_2 \setminus \alpha \\
\\
(\mu.p)[\phi] &\simeq \phi(\mu).p[\phi] \\
(p_1 + p_2)[\phi] &\simeq p_1[\phi] \; + \; p_2[\phi] \\
\\
\mathbf{nil} \setminus \alpha &\simeq \mathbf{nil}[\phi] \simeq \mathbf{nil} \mid \mathbf{nil} \simeq \mathbf{nil} \rfloor p \simeq \mathbf{nil} \,\|\, p \simeq p \|\, \mathbf{nil} \simeq \mathbf{nil}
\end{aligned}$$

□

From the previous theorem, it follows that every finite CCS agent can be equivalently written using action prefix, sum and **nil**. Then, the axioms that characterize the strong bisimilarity relation are the following:

$$\begin{aligned}
p + \mathbf{nil} &= p \\
p_1 + p_2 &= p_2 + p_1 \\
p_1 + (p_2 + p_3) &= (p_1 + p_2) + p_3 \\
p + p &= p
\end{aligned}$$

Note that the axioms simply assert that processes with sum define an idempotent, commutative monoid whose neutral element is **nil**.

10.7. Weak Semantics of CCS

Let us now see an example that illustrates the limits of strong bisimilarity as a behavioural equivalence between agents.

Example 10.27
Let p and q be the following CCS agents:

$$p = \tau.\,\mathbf{nil} \qquad q = \mathbf{nil}$$

Obviously the two agents are distinguished by the (invisible) action τ. So they are not bisimilar, but, since we consider τ as an internal action, not visible from outside of the system, we have that, according to the observable behaviours, they should not be distinguished.

The above example shows that strong bisimilarity is not abstract enough. So we could think to abstract away from the invisible transition by defining a new relation. This relation is called *weak bisimilarity*. We start by defining a new, more abstract, LTS.

10.7.1. Weak Bisimilarity

Definition 10.28 (Weak transitions)
We let $\Longrightarrow$ be the weak transition relation *on the set of states of an LTS defined as follows:*

$$p \overset{\tau}{\Rightarrow} q \quad \textit{iff} \quad p \overset{\tau}{\rightarrow} \ldots \overset{\tau}{\rightarrow} q \;\vee\; p = q$$
$$p \overset{\lambda}{\Rightarrow} q \quad \textit{iff} \quad p \overset{\tau}{\Rightarrow} p' \overset{\lambda}{\rightarrow} q' \overset{\tau}{\Rightarrow} q$$

Note that $p \overset{\tau}{\Rightarrow} q$ means that q can be reached from p via a possibly empty sequence of τ-transitions, i.e., $\overset{\tau}{\Rightarrow}$ coincides with the reflexive and transitive closure $(\overset{\tau}{\rightarrow})^*$ of invisible transition $\overset{\tau}{\rightarrow}$, while $p \overset{\lambda}{\Rightarrow} q$ requires the execution of one visible transition (the one labelled with λ).

Now, as done for the strong bisimilarity, we define a function $\Psi : \mathscr{P}(P \times P) \rightarrow \mathscr{P}(P \times P)$ which takes a relation on P and returns a another relation (if it exists) by exploiting weak transitions:

$$p\ \Psi(R)\ q \overset{\text{def}}{=} \begin{cases} p \overset{\mu}{\rightarrow} p' \text{ then } \exists q'.\ q \overset{\mu}{\Rightarrow} q' & \text{and} \quad p'\ R\ q' \\ q \overset{\mu}{\rightarrow} q' \text{ then } \exists p'.\ p \overset{\mu}{\Rightarrow} p' & \text{and} \quad p'\ R\ q' \end{cases}$$

And we define the *weak bisimilarity* as follows:

$$p \approx q \;\text{ iff }\; \exists R.\ p\ R\ q \wedge \Psi(R) \sqsubseteq R$$

This relation seems to improve the notion of equivalence w.r.t. $\simeq$, because $\approx$ abstracts away from the invisible transitions as we required. Unfortunately, there are two problems with this relation. First, the $\Rightarrow$ LTS is infinite branching also for guarded terms (consider e.g. *rec x.*$(\tau.x|\alpha.nil)$, analogous to the agent discussed in example 10.15). Thus function Ψ is not continuous, and the minimal fixpoint, which exists anyway, cannot be reached in general with an ω-chain of approximations. Second, and much worse, weak bisimilarity is not a congruence with respect to the + operator, as the following example shows. As a (minor) consequence, weak bisimilarity, differently than strong bisimilarity, cannot be axiomatized.

Example 10.29
Let p and q be the following CCS agents:

$$p = \alpha.\,\mathbf{nil} \qquad q = \tau.\alpha.\,\mathbf{nil}$$

Obviously for the weak equivalence we have $p \approx q$, since their behaviours differ only by the ability to perform an invisible action τ. Now we define the following context:

$$C[\_] = \_ + \beta.\,\mathbf{nil}$$

Then by embedding p and q within the context $C[\_]$ we obtain:

$$\alpha.\,\mathbf{nil} + \beta.\,\mathbf{nil} \not\approx \tau.\alpha.\,\mathbf{nil} + \beta.\,\mathbf{nil}$$

In fact $C[q]$ can perform a τ-transition and become $\alpha.\,\mathbf{nil}$, while $C[p]$ has only one invisible weak transition that can be used to match such a step, but such weak transition is the idle step $C[p] \stackrel{\tau}{\Rightarrow} C[p]$ and $C[p]$ is clearly not equivalent to $\alpha.\,\mathbf{nil}$ (because the former can perform a β-transition that the latter cannot simulate). This phenomenon is due to the fact that τ-transitions are not observable but can be used to discard some non-deterministic choices. While quite unpleasant, the above fact is not in any way due to a CCS weakness, or misrepresentation of reality, but rather enlightens a general property of nondeterministic choice in systems represented as black boxes.

10.7.2. Weak Observational Congruence

As shown by the Example 10.29, weak bisimilarity is not a congruence relation. In this section we will show a possible (partial) solution. Since weak bisimilarity equivalence is a congruence for all operators except sum, to fix our problem it is enough to impose closure for all sum contexts.

Let us consider the Example 10.29, where the execution of a τ-transition forces the system to make a choice which is invisible to an external observer. In order to make this kind of choices observable we can define the relation $\cong$ as follows:

$$p \cong q \;\text{ iff }\; p \approx q \;\wedge\; \forall r.\; p + r \approx q + r$$

This relation, called *weak observational congruence*, can be defined directly as:

$$\begin{array}{lllll} p \cong q & \text{iff} \quad p \xrightarrow{\tau} p' & \text{implies} \quad q \xrightarrow{\tau} \stackrel{\tau}{\Longrightarrow} q' & \text{and} \quad p' \approx q' \\ & \text{iff} \quad p \xrightarrow{\lambda} p' & \text{implies} \quad q \stackrel{\lambda}{\Longrightarrow} q' & \text{and} \quad p' \approx q' \\ & \text{(and vice versa)} & & \end{array}$$

As we can see we avoided the possibility to stop after the execution of an internal action. Notice however that this is not a fixpoint definition, since $\cong$ is simply defined in terms of $\approx$. This relation is a congruence but as we can see in the following example it is not a bisimulation according to Ψ, namely $\Psi(\cong) \neq \cong$.

Example 10.30
Let p and q defined as follows:

$$p = \alpha.\tau.\beta.\,\mathbf{nil} \quad \textit{and} \quad q = \alpha.\beta.\,\mathbf{nil}$$

we have:

$$p \cong q$$

but if Alice plays α on p, Bob has no chance of playing α and of reaching a state in relation $\cong$:

$$p' = \tau.\beta.\,\mathbf{nil} \quad \textit{and} \quad q' = \beta.\,\mathbf{nil}$$

since $p' \not\cong q'$. Thus $\cong$ is not a fixpoint of Ψ.

It is possible to prove that the equivalence relation $\cong$ can be axiomatized by adding to the axioms for strong bisimilarity the following three Milner's τ laws:

$$p + \tau.p = \tau.p$$
$$\mu.(p + \tau.q) = \mu.(p + \tau.q) + \mu.q$$
$$\mu.\tau.p = \mu.p$$

10.7.3. Dynamic Bisimilarity

As shown by the Example 10.30 the observational congruence is not a bisimulation. In this section we present the largest relation which is at the same time a congruence and a Ψ-bisimulation. It is called *dynamic bisimilarity* and was introduced by Vladimiro Sassone.

We define the dynamic bisimilarity $\approx_d$ as the largest relation that satisfies:

$$p \approx_d q \quad \text{implies} \quad \forall C.\ C[p]\ \Psi(\approx_d)\ C[q]$$

In this case, at every step we close the relation by comparing the behaviour w.r.t. any possible embedding context. In terms of game theory this definition can be viewed as "at each turn Alice is also allowed to insert both agents into the same context in order to win."

As for the observational congruence, we can define the dynamic bisimilarity as follows:

$$p\ \Theta(R)\ q \stackrel{\text{def}}{=} \begin{cases} p \xrightarrow{\tau} p' \text{ then } \exists q'.\ q \xrightarrow{\tau} \stackrel{\tau}{\Longrightarrow} q' \quad \text{and} \quad p'\ R\ q' \\ p \xrightarrow{\lambda} p' \text{ then } \exists q'.\ q \stackrel{\tau}{\Longrightarrow} \xrightarrow{\lambda} \stackrel{\tau}{\Longrightarrow} q' \quad \text{and} \quad p'\ R\ q' \\ \text{(and vice versa)} \end{cases}$$

Then, R is a *dynamic bisimulation* if $\Theta(R) \sqsubseteq R$, and the dynamic bisimilarity is obtained by letting:

$$\approx_d = \bigsqcap_{R \subseteq \Theta(R)} R$$

Example 10.31
Let p and q be defined as in the Example 10.30.

$$p = \alpha.\tau.\beta.\,\mathbf{nil} \quad \textit{and} \quad q = \alpha.\beta.\,\mathbf{nil}$$

$$p' = \tau.\beta.\,\mathbf{nil} \quad \textit{and} \quad q' = \beta.\,\mathbf{nil}$$

we have:

$$p \not\approx_d q \quad \textit{and} \quad p' \not\approx_d q'$$

As for the observational congruence we can finitely axiomatize the dynamic bisimilarity. The axiomatization of $\approx_d$ is obtained by omitting the third Milner's τ law as follows:

$$p + \tau p = \tau p$$
$$\mu(p + \tau q) = \mu(p + \tau q) + \mu q$$

Part IV.

Temporal and Modal Logic

11. Temporal Logic and μ-Calculus

As we have discussed in the previous chapter (see Section 10.5) modal logic is a powerful tool that allows to check some behavioral properties of systems. In Section 10.5 the focus was on Hennessy-Milner logic, whose main limitation is due to its finitary structure: only local properties can be investigated. In this chapter we show some extensions of Hennessy-Milner logic that increase the expressiveness of the formulas. The most powerful language that we will present is the *μ-Calculus*. It allows to express complex constraints about the infinite behaviour of our systems.

Classically, we can divide the properties to be investigated in three categories:

- *safety*: if the property expresses the fact that something bad will not happen.
- *liveness*: if the property expresses the fact that something good will happen.
- *fairness*: if the property expresses the fact that something good will happen infinitely many times.

11.1. Temporal Logic

The first step in extending modal logic is to introduce the concept of time in our models. This will extend the expressiveness of modal logic, making it able to talk about concepts like "ever","never" or "sometimes". In order to represent the concept of time in our logics we have to represent it in a mathematical fashion. In our discussion we assume that the time is discrete and infinite.

While temporal logic shares similarities with HM-logic, note that:

- temporal logic is based on a set of *atomic propositions* whose validity is associated with a set of states, i.e., the observations are taken on states and not on (actions labeling the) arcs;
- temporal operators allows to look further than the "next" operator of HML;
- as we will see, the choice of representing the time as linear (linear temporal logic) or as tree (computation tree logic) will lead to different types of logic, that roughly correspond to the trace semantic view vs the bisimulation semantics view.

11.1.1. Linear Temporal Logic

In the case of *Linear Temporal Logic* (LTL) the time is represented as a line. This means that the evolutions of the system are linear, they proceed from a state to another without making any choice. The formulas of LTL are based on a set of *atomic propositions*, which can be composed using the classical logic operators together with the following temporal operators:

- O: is called *next* operator. The formula $O\phi$ means that ϕ is true in the next state (i.e., in the next instant of time). Some literature uses X or N in place of O.
- F: is called *finally* operator. The formula $F\phi$ means that ϕ is true sometime in the future.
- G: The formula $G\phi$ means that ϕ is always (*globally*) valid in the future.
- U: is called *until* operator. The formula $\phi_1 U \phi_2$ means that ϕ_1 is true until the first time that ϕ_2 is true.

In order to represent the state of the system while the time elapses we introduce the following mathematical structure.

Definition 11.1 (Linear structure)
Let P be a set of atomic propositions and $S : P \to 2^\omega$ be a function from the atomic propositions to subsets of natural numbers defined as follows:

$$\forall p \in P.\, S(p) = \{x \in \omega \mid x \text{ satisfies } p\}$$

Then we call the pair (S, P) a linear structure.

In a linear structure, the natural numbers $0, 1, 2 \ldots$ represent the time instants, and the states in them, and S represents for every predicate the states where it holds, or, alternatively, it represents for every state the predicates which it satisfies.The operators of LTL allows to quantify (existentially and universally) w.r.t. the traversed states. To define the satisfaction relation, we need to check properties on future states, like some sort of "time travel". To this aim we define the following *shifting* operation on S:

$$\forall i \in \omega\, \forall p \in P.\, S^i(p) = \{x - i \mid x \geq i \ \wedge \ x \in S(p)\}$$

As done for the HM-logic, we define the satisfaction operator $\models$ as follows:

- $S \models p$ if $0 \in S(p)$
- $S \models true$
- $S \models \neg\phi$ if it is not true that $S \models \phi$
- $S \models \phi_1 \wedge \phi_2$ if $S \models \phi_1$ and $S \models \phi_2$
- $S \models \phi_1 \vee \phi_2$ if $S \models \phi_1$ or $S \models \phi_2$
- $S \models O\phi$ if $S^1 \models \phi$
- $S \models F\phi$ if $\exists i \in \omega$ such that $S^i \models \phi$
- $S \models G\phi$ if $\forall i \in \omega$ it holds $S^i \models \phi$
- $S \models \phi_1 U \phi_2$ if $\exists i \in \omega$ such that $S^i \models \phi_2$ and $\forall j < i\ S^j \models \phi_1$

We say that an LTL formula ϕ is satisfiable if there is some computation S such that $S \models \phi$.

From the satisfaction relation it is easy to check that the operators F and G can be expressed in terms of the until operator as follows:

$$\begin{aligned} F\phi &\equiv true\ U\ \phi \\ G\phi &\equiv \neg F \neg\phi \equiv \neg(true\ U\ \neg\phi) \end{aligned}$$

We now show some examples that illustrate how powerful the LTL is.

Example 11.2

- *$G\neg p$: expresses the fact that p will never happen, so it is a safety property.*
- *$p \to Fq \equiv \neg p \vee Fq$: expresses the fact that if p happens then also q will happen sometime in the future.*
- *GFp: expresses the fact that p happens infinitely many times in the future, so it is a fairness property.*
- *FGp: expresses the fact that p will always hold some time in the future.*
- *$G(request \to (request\ U\ grant))$: expresses the fact that whenever a request is made it holds continuously until it is eventually granted.*

11.1.2. Computation Tree Logic

In this section we introduce CTL and CTL^* two logics which use trees as models of the time. CTL and CTL^* extend LTL with two operators which allows to express properties on paths over trees. The difference between CTL and CTL^* is that the former is a restricted version of the latter. So we start by introducing CTL^*.

We introduce two new operators on paths:

- E: the formula $E\phi$ (read "possibly ϕ") means that there *exists* some path that satisfies ϕ;
- A: the formula $A\phi$ (read "inevitably ϕ") means that each path of the tree satisfies ϕ, i.e., that ϕ is satisfied along *all* paths.

This time the state of the system is represented by using infinite trees as follows.

Definition 11.3 (Infinite tree)
Let $T = (V, \rightarrow)$ be a tree, with V the set of nodes, v_0 the root and $\rightarrow \subseteq V \times V$ the parent-child relation. We say that T is an infinite tree *if the following holds:*

$$\rightarrow \text{ is total on } V, \text{ namely } \forall v \in V\ \exists w \in V.\ v \rightarrow w$$

Definition 11.4 (Branching structure)
Let P be a set of atomic propositions, $T = (V, \rightarrow)$ be an infinite tree and $S : P \rightarrow 2^V$ be a function from the atomic propositions to subsets of nodes of V defined as follows:

$$\forall p \in P.\ S(p) = \{x \in V \mid x \text{ satisfies } p\}$$

Then we call (T, S, P) a branching structure.

We are interested in infinite paths on trees.

Definition 11.5 (Infinite paths)
Let $(V, \rightarrow)$ be an infinite tree and $\pi = v_0, v_1, \ldots, v_n, \ldots$ be an infinite sequence of nodes in V. We say that π is an infinite path *over $(V, \rightarrow)$ iff*

$$\forall i \in \omega.\ v_i \rightarrow v_{i+1}$$

As for the linear case, we need a shifting operators on path. So for $\pi = v_0, v_1, \ldots, v_n, \ldots$ we let π^i be defined as follows:

$$\forall i \in \omega.\ \pi^i = v_i, v_{i+1}, \ldots$$

Let (T, S, P) be a branching structure and $\pi = v_0, v_1, \ldots, v_n, \ldots$ be an infinite path. We define the $\models$ relation as follows:

state operators:

- $S, \pi \models p$ if $v_0 \in S(p)$
- $S, \pi \models \neg\phi$ if it is not true that $S, \pi \models \phi$
- $S, \pi \models \phi_1 \wedge \phi_2$ if $S, \pi \models \phi_1$ and $S, \pi \models \phi_2$

- $S, \pi \models \phi_1 \vee \phi_2$ if $S, \pi \models \phi_1$ or $S, \pi \models \phi_2$
- $S, \pi \models O\phi$ if $S, \pi^1 \models \phi$
- $S, \pi \models F\phi$ if $\exists i \in \omega$ such that $S, \pi^i \models \phi$
- $S, \pi \models G\phi$ if $\forall i \in \omega$ it holds $S, \pi^i \models \phi$
- $S, \pi \models \phi_1 U \phi_2$ if $\exists i \in \omega$ such that $S, \pi^i \models \phi_2$ and for all $j < i$ $S, \pi^j \models \phi_1$

path operators

- $S, \pi \models E\phi$ if there exists $\pi_1 = v_0, v'_1, \ldots, v'_n, \ldots$ such that $S, \pi_1 \models \phi$
- $S, \pi \models A\phi$ if for all paths $\pi_1 = v_0, v'_1, \ldots, v'_n, \ldots$ we have $S, \pi_1 \models \phi$

Let us see some examples.

Example 11.6

- *EOp: it is the same of the next operator $\Diamond$ in modal logic.*
- *AGp: expresses the fact that p happens in all reachable states.*
- *EFp: expresses the fact that p happens in some reachable state.*
- *AFp: expresses the fact that on every path there exists a state where p holds.*
- *$E(pUq)$: expresses the fact that there exists a path where p holds until q.*
- *$AGEFp$: in every future exists a successive future where p holds.*

The formulas of CTL are obtained by restricting CTL^*: a CTL^* formula is a CTL formula if the followings hold:

- A and E appear only immediately before a linear operator (i.e., F,G,U and O).
- each linear operator appears immediately after a quantifier (i.e., A and E).

It is evident that CTL and LTL are both subsets of CTL^*, but they are not equivalent to each other. Without going into the detail, we mention that:

- no CTL formula is equivalent to the LTL formula $F(Gp)$;
- no LTL formula is equivalent to the CTL formula $AG(p \rightarrow (EOq \wedge EO\neg q))$

Finally, we note that all CTL formulas can be written in terms of the minimal set of operators *true*, $\neg$, $\vee$, EG, EU, EO. In fact, for the remaining operators we have the following logical equivalences:

$$\begin{aligned}
EF\phi &\equiv E(true\ U\ \phi) \\
AO\phi &\equiv \neg(EO\neg\phi) \\
AG\phi &\equiv \neg(EF\neg\phi) \equiv \neg E(true\ U\ \neg\phi) \\
AF\phi &\equiv A(true\ U\ \phi) \equiv \neg(EG\neg\phi) \\
A(\phi\ U\ \varphi) &\equiv \neg(E(\neg\varphi\ U\ \neg(\phi \vee \varphi)) \vee EG\neg\varphi)
\end{aligned}$$

11.2. μ-Calculus

Now we introduce the μ-calculus. The idea is to add the least and greatest fixpoint operators to modal logic. This fits nicely with the fact that many interesting properties can be conveniently expressed as fixpoints. The two operators that we introduce are the following:

- $\mu x.\phi$ is the least fixpoint of ϕ.
- $\nu x.\phi$ is the greatest fixpoint of ϕ.

Note that, in order to apply the fixpoint operators we require that ϕ is monotone, this means that any occurrence of x in ϕ must be preceded by an even number of negations. The μ-calculus is interpreted on LTSs.

Let $(V, \rightarrow)$ be an LTS, X be the set of predicate variables and P be a set of predicates, we introduce a function $\rho : P \cup X \rightarrow 2^V$ which associates to each predicate and each free variable a subset of vertices. Then we define the denotational semantics of μ-calculus which maps each predicate to the subset of states in which it holds as follows:

$$
\begin{aligned}
[\![x]\!]\rho &= \rho x \\
[\![p]\!]\rho &= \rho p \\
[\![\phi_1 \wedge \phi_2]\!]\rho &= [\![\phi_1]\!]\rho \cap [\![\phi_2]\!]\rho \\
[\![\phi_1 \vee \phi_2]\!]\rho &= [\![\phi_1]\!]\rho \cup [\![\phi_2]\!]\rho \\
[\![\neg\phi]\!]\rho &= V \setminus [\![\phi]\!]\rho \\
[\![true]\!]\rho &= V \\
[\![false]\!]\rho &= \varnothing \\
[\![\Diamond\phi]\!]\rho &= \{\, v \mid \exists v'.v \rightarrow v' \ \wedge\ v' \in [\![\phi]\!]\rho \,\} \\
[\![\Box\phi]\!]\rho &= \{\, v \mid \forall v'.v \rightarrow v' \Rightarrow v' \in [\![\phi]\!]\rho \,\} \\
[\![\mu x.\phi]\!]\rho &= \mathrm{fix}\, \lambda S.\, [\![\phi]\!]\rho[^S/_x] \\
[\![\nu x.\phi]\!]\rho &= \mathrm{Fix}\, \lambda S.\, [\![\phi]\!]\rho[^S/_x]
\end{aligned}
$$

Example 11.7

- $[\![\mu x.x]\!]\rho = \varnothing$
- $[\![\nu x.x]\!]\rho = V$
- $[\![\mu x.\Diamond x]\!]\rho = \mathrm{fix}\, \lambda S.\{v \mid \exists v'.\ v \rightarrow v' \wedge v' \in S\}$
  we have:
  $$S_0 = \varnothing \qquad S_1 = \{v \mid \exists v'.\ v \rightarrow v' \wedge v' \in \varnothing\} = \varnothing \quad \textit{(fixpoint reached)}$$
- $[\![\mu x.\Box x]\!]\rho = \mathrm{fix}\, \lambda S.\{v \mid \forall v'.\ v \rightarrow v' \Rightarrow v' \in S\}$
  we have:

  $S_0 = \varnothing \qquad S_1 = \{v \mid \forall v'.\ v \rightarrow v' \Rightarrow v' \in \varnothing\} = \{v \mid v \nrightarrow\}$ *the set of vertices with no outgoing arcs*

  $S_2 = \{v \mid \forall v'.\ v \rightarrow v' \Rightarrow v' \in S_1\} =$ *the set of vertices with outgoing paths of length at most* 1

  $S_n = \{v \mid \forall v'.\ v \rightarrow v' \Rightarrow v' \in S_{n-1}\} =$ *the set of vertices with outgoing paths of length at most* $n-1$

  $\bigcup_{i \in \omega} S_i =$ *vertices with only finite outgoing paths*
- $[\![\nu x.\Box x]\!]\rho = \mathrm{Fix}\, \lambda S.\{v \mid \forall v', v \rightarrow v', v' \in S\}$
  we have:
  $$S_0 = V \qquad S_1 = \{v \mid \forall v'.\ v \rightarrow v' \Rightarrow v' \in V\} = V \quad \textit{(fixpoint reached)}$$

- $[\![\mu x.p \vee \Diamond x]\!]\rho = \text{fix}\, \lambda S.\rho p \cup \{v \mid \exists v'.v \to v' \wedge v' \in S\}$ *(similar to EFp, meaning some node in ρp is reachable)*
  we have:

  $S_0 = \varnothing$ $\quad S_1 = \rho p$ $\quad S_2 = \rho p \cup \{v \mid \exists v'.\ v \to v' \wedge v' \in \rho p\} = \rho p$ *is reachable in at most one step*

  $S_n = \rho p$ *is reachable in at most* $n-1$ *steps* $\quad \bigcup_{i \in \omega} S_i = \rho p$ *is reachable (in any number of steps)*

- $[\![\nu x.\mu y.(p \wedge \Diamond x) \vee \Diamond y]\!]\rho$ *(corresponds to $EGFp$)*
  start a path, $\mu y.(p \wedge \Diamond x) \vee \Diamond y$ means that after a finite number of steps you find a vertex where both (1) p holds and (2) you can reach a vertex where the property recursively holds.

- $[\![\mu x.(p \wedge \Box x \wedge \Diamond x) \vee q]\!]\rho = \text{fix}\, \lambda S.(\rho p \cap \{v \mid \forall v'.\ v \to v' \Rightarrow v' \in S\} \cap \{v \mid \exists v'.\ v \to v' \wedge v' \in S\}) \cup \rho q$ *(corresponds to $ApUq$)*

  Note that in this case the $\Diamond x$ is necessary in order to ensure that the state is not a deadlock one.

- $[\![\mu x.(p \wedge \Diamond x) \vee q]\!]\rho = \text{fix}\, \lambda S.(\rho p \cap \{v \mid \exists v'.\ v \to v' \wedge v' \in S\}) \cup \rho q$ *(corresponds to $EpUq$)*

11.3. Model Checking

The problem of model checking consists in the, possibly automatic, verification of whether a given model of a system meets or not a given logic specification of the properties the system should satisfy, like absence of deadlocks.

The main ingredients of model checking are:

- an LTS (the model) and a vertex (the initial state);
- a formula (in temporal or modal logic) you want to check (for that state in the model)

The result of model checking should be either a positive answer (the given state in the model satisfies the formula) or some counterexample explaining one possible reason why the formula is not satisfied.

In the case of concurrent systems, the LTS is often given implicitly, as the one associated with a term of some process algebra, because in this way the structure of the system is handled more conveniently. However the size of the actual translation can explode even if the system is finite state. For example, let $p_i = \alpha_i.\,\mathbf{nil}$ for $i = 1, .., n$ and take the CCS process $s = p_1 \mid p_2 \mid \cdots \mid p_n$: the number of reachable states of the resulting model is 2^n.

One possibility to tackle the state explosion problem is to minimize the system according to some suitable equivalence. Note that minimization can take place also while combining subprocesses and not just at the end. Of course, this technique is viable only if the minimization is related to an equivalence relation that respects the properties to be checked. For example, the μ-calculus is invariant w.r.t. bisimulation, thus we can minimize CCS processes up to bisimilarity before model checking them.

In model checking algorithms, it is often convenient to proceed by evaluating formulas with the aid of dynamic programming. The idea is to work in a bottom-up fashion: starting from the atomic predicates that appear in the formula, we mark all the states with the sub formulas they satisfy. When a variable is encountered, a separate activation of the procedure is allocated for computing the fixpoint of the corresponding recursive definition. The complexity becomes very large in the case of formulas that involve many least and greatest fix points in alternation.

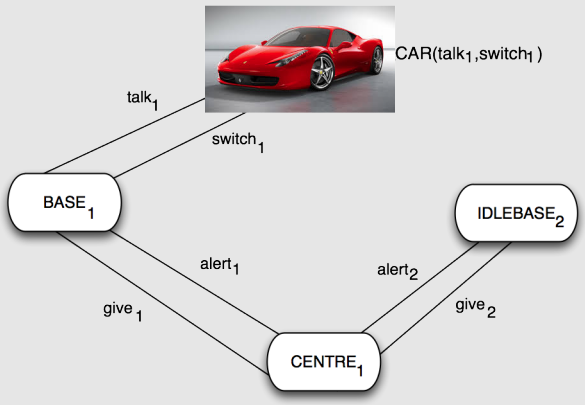
Part V.

π-calculus

12. π-Calculus

The structures of today's communication systems are not statically defined, but they change continuously according to the needs of the users. The CCS calculus we saw in chapter 10 is unsuitable for modeling such systems, since its communication structure (the channels) cannot evolve dynamically. In this chapter we present the π-calculus, an extension of CCS introduced by Robin Milner, Joachim Parrow and David Walker in 1989 which allows to model mobile systems. The main feature of the π-calculus is its ability of creating new channels and of sending them in messages allowing agents to change their connections. Let us start with an example which illustrates how the π-calculus can formalize a mobile telephone system.

Example 12.1 (Mobile Phones)
The following figure which represents a mobile phone network: while the car travels, the phone can communicate with different bases in the city, but just one at a time, typically the closest to its position. The communication centre decides when the base must be changed and then the channel for accessing the new base is sent to the car through the switch channel.



As for CCS, also in this case we describe agent behaviour by defining the reachable states:

$$CAR(talk, switch) \stackrel{\text{def}}{=} \overline{talk}.CAR(talk, switch) + switch(talk', switch').CAR(talk', switch')$$

A car can talk on the channel assigned by the communication centre (action $\overline{talk}$). Alternatively the car can receive (action $switch(talk', switch')$) a new pair of channels ($talk'$ and $switch'$) and change the base to which it is connected.

$$BAS\,E_i \stackrel{\text{def}}{=} BAS\,E(talk_i, switch_i, give_i, alert_i) \stackrel{\text{def}}{=}$$
$$talk_i.BAS\,E_i + give_i(talk', switch').\overline{switch_i}(talk', switch').IDLEBAS\,E_i$$

$$IDLEBAS\,E_i \stackrel{\text{def}}{=} IDLEBAS\,E(talk_i, switch_i, give_i, alert_i) \stackrel{\text{def}}{=} alert_i.BAS\,E_i$$

A generic base can be in two possible states: $BAS\,E$ or $IDLEBAS\,E$. In the first case the base is connected to the car, so either the phone can talk or the base can receive two channels from the centre and send them to the car for allowing it to change base. In the second case the base is idle, so it can only be awakened by the communication centre.

$$CENTRE_1 \stackrel{\text{def}}{=} CENTRE_1(give_1, alert_1, give_2, alert_2) = \overline{give_1}(talk_2, switch_2).\overline{alert_2}.CENTRE_2$$
$$CENTRE_2 \stackrel{\text{def}}{=} CENTRE_2(give_1, alert_1, give_2, alert_2) = \overline{give_2}(talk_1, switch_1).\overline{alert_1}.CENTRE_1$$

The communication centre can be in different states according to which base is active. In the example there are only two possible states for the communication centre ($CENTRE_1$ and $CENTRE_2$), because only two bases are considered.

$$SYSTEM_1 \stackrel{\text{def}}{=} (CAR(talk_1, switch_1)|BASE_1|IDLEBASE_2|CENTRE_1)$$

Finally we have the process which represents the entire system in the state where the first car is talking.

Example 12.2 (Secret Channel via Trusted Server)
As another example, consider two processes Alice (A) and Bob (B) that want to establish a secret channel using a trusted server (S) with which they already have trustworthy communication link c_{AS} (for Alice to send private messages to the server) and c_{SB} (for the server to send private messages to Bob). The system can be represented by the expression:

$$Sys \stackrel{\text{def}}{=} (c_{AS})(c_{BS})(A|S|B)$$

where the restrictions *(c_{AS}) and (c_{BS}) guarantees that the link c_{AS} and c_{SB} are not visible from the environment and where the processes A, S and B are specified as follows:*

$$\begin{aligned} A &\stackrel{\text{def}}{=} (c_{AB})\bar{c}_{AS}c_{AB}.\bar{c}_{AB}m.p_A \\ S &\stackrel{\text{def}}{=} !c_{AS}(x).\bar{c}_{SB}x.\,\mathbf{nil} \\ B &\stackrel{\text{def}}{=} c_{SB}(y).y(m).q_B \end{aligned}$$

Restriction (x) is similar to the CCS operator $\backslash x$, with the important difference that in π-calculus the scope of the restriction can change as the process evolves. Alice defines a private name c_{AB} that wants to use for communicating with B, then Alice sends the name c_{AB} to the trusted server over their private shared link c_{AS} and finally sends the message m on the channel c_{AB} and continues as p_A. The server continuously wait fro messages from Alice on channel s_{AS} and forwards the content to Bob. Here the replication operator ! allows to serve multiple requests from Alice. Bob waits to receive the name y from the server over the channel c_{SB} and then uses y to input the message from Alice and continue as q_B (which can now use both y and m).

12.1. Syntax of π-calculus

The π-calculus has been introduced to model communicating systems where channel names, representing addresses and links, can be created and forwarded. To this aim we rely on a set of channel names $x, y, z, ...$ and extend the CCS actions with the ability to send and receive channel names. In these notes we present the monadic version of the calculus, namely the version where names can be sent only one at a time. We introduce the syntax, with productions for processes and for actions.

$$\begin{aligned} p &::= \mathbf{nil} \mid \alpha.p \mid [x = y]p \mid p + p \mid p|p \mid (y)p \mid !p \\ \alpha &::= \tau \mid x(y) \mid \bar{x}y \end{aligned}$$

The meaning of the operators for building π-calculus processes is the following:

- **nil** is the inactive agent.
- $\alpha.p$ is an agent which can perform an action α and then act like p.
- $[x = y]p$ is the conditional process, which acts like p if $x = y$, while otherwise the process remains blocked.
- $p + q$ is the non-deterministic choice between two processes.
- $p|q$ is the parallel composition of two processes.
- $(y)p$ denotes the restriction of the channel y, which makes the name y private in p.
- $!p$ is a replicated process: it behaves as if an unbounded number of concurrent occurrences of p were available in parallel.

The meaning of the actions is the following:

- τ as usual is the invisible action.
- $x(y)$ is the input on channel x, the received value would be stored in y.
- $\overline{x}y$ is the output on channel x of the name y.

In the above case, we call x the *subject* of the communication (i.e., the channel name where the communication takes place) and y the *object* of the communication (i.e., the channel name that is transmitted or received). As in the λ-calculus, in the π-calculus we have *bound* and *free* occurrence of names. The bounding operators of π-calculus are input and restriction:

- $x(y).p$ (name y is bound in p).
- $(y)p$ (name y is bound in p).

On the contrary, the output prefix is not binding, i.e., if we take the process $\overline{x}y.p$ then the name y is said to be *free* in p. Note that for both $x(y).p$ and $\overline{x}y.p$ the name x is free in p. Moreover we define the *name set* of α as follows:

$$n(\alpha) = fn(\alpha) \cup bn(\alpha)$$

Unlike for CCS, the restriction operator $(y)p$ does not bind statically the scope of y to coincide with p. In fact in the π-calculus channel names are values, so the process p can send the name y to another process which thus becomes part of the scope of y. The possibility to enlarge the scope of a restricted name is a very useful feature of the π-calculus, called *extrusion*, which allows to modify the structure of private communications between agents.

12.2. Operational Semantics of π-calculus

Likewise CCS, we define the operational semantics by using a rule system, where well formed formulas are triples $p \xrightarrow{\alpha} q$ as for CCS. The actions α that can label the transitions are: (i) the silent action τ; (ii) the input $x(y)$ of name y on channel x; (iii) the free output $\bar{x}y$ of name y on channel x; (iv) the bound output $\bar{x}(y)$ of a previously restricted name y on channel x. The definition of free names $fn(\cdot)$, bound names $bn(\cdot)$ and names $n(\cdot)$ are extended to labels by letting:

- $fn(\tau) = \emptyset$, $fn(x(y)) = fn(\bar{x}(y)) = \{x\}$, and $fn(\bar{x}y) = \{x, y\}$;
- $bn(\tau) = bn(\bar{x}y) = \emptyset$ and $bn(x(y)) = fn(\bar{x}(y)) = \{y\}$;
- $n(\alpha) = fn(\alpha) \cup bn(\alpha)$.

$$\text{(Tau)}\ \frac{}{\tau.p \xrightarrow{\tau} p}$$

The rule (Tau) allows to perform invisible actions.

$$\text{(Out)}\ \frac{}{\overline{x}\,y.p \xrightarrow{\overline{x}\,y} p}$$

As we said the π-calculus processes can exchange messages which can contain information (i.e., channel names). The rule (Out) allows p to send the name y on the channel x.

$$\text{(In)}\ \frac{}{x(y).p \xrightarrow{x(w)} p\{^{w}/_{y}\}}\ w \notin fn((y)p)$$

The rule (In) allows to receive in input over x some channel name. The received name w is bound to the name y in the process p. In order to avoid name conflicts, we assume w does not appear as a free name in $(y)p$, i.e., the transition is defined only when w is *fresh*.

$$\text{(SumL)}\ \frac{p \xrightarrow{\alpha} p'}{p + q \xrightarrow{\alpha} p'} \qquad \text{(SumR)}\ \frac{q \xrightarrow{\alpha} q'}{p + q \xrightarrow{\alpha} q'}$$

The rules (SumL) and (SumR) allow the system $p + q$ to behave as p or q.

$$\text{(Match)}\ \frac{p \xrightarrow{\alpha} p'}{[x = x]p \xrightarrow{\alpha} p'}$$

The rule (Match) allows to check the condition between square bracket and unblock the process p. If the matching condition is not satisfied we can not continue the execution.

$$\text{(ParL)}\ \frac{p \xrightarrow{\alpha} p'}{p|q \xrightarrow{\alpha} p'|q}\ bn(\alpha) \cap fn(q) = \varnothing \qquad \text{(ParR)}\ \frac{q \xrightarrow{\alpha} q'}{p|q \xrightarrow{\alpha} p|q'}\ bn(\alpha) \cap fn(p) = \varnothing$$

As for CCS the two rules (ParL) and (ParR) allow the interleaved execution of two π-calculus agents. The side conditions guarantee that the bound names in α (if any) are fresh w.r.t. the idle process. Notice that if we assume that the bound names of α are fresh wrt. the premise of the rule, thanks to the side condition we can conclude that they are fresh also wrt. the consequence.

$$\text{(ComL)}\ \frac{p \xrightarrow{\overline{x}\,z} p' \quad q \xrightarrow{x(y)} q'}{p|q \xrightarrow{\tau} p'|(q'\{^{z}/_{y}\})} \qquad \text{(ComR)}\ \frac{p \xrightarrow{x(y)} p' \quad q \xrightarrow{\overline{x}\,z} q'}{p|q \xrightarrow{\tau} p'\{^{z}/_{y}\}|q'}$$

The rules (ComL) and (ComR) allow the synchronization of two parallel process. The formal name y is replaced with the actual name z in the continuation of the receiver.

$$\text{(Res)}\ \frac{p \xrightarrow{\alpha} p'}{(y)p \xrightarrow{\alpha} (y)p'}\ y \notin n(\alpha)$$

The rule (Res) expresses the fact that if a name y is restricted on top of the process p, then any action which does not involve y can be performed by p.

Now we present the most important rules of π-calculus Open and Close, dealing with *scope extrusion* of channel names. Rule Open *publishes*, i.e. makes free, a private channel name, while rule Close restricts again the name, but with a broader scope.

$$\text{(Open)}\ \frac{p \xrightarrow{\overline{x}\,y} p'}{(y)p \xrightarrow{\overline{x}(w)} p'\{^{w}/_{y}\}}\ y \neq x \quad w \notin fn((y)p)$$

The rule (Open) publishes the private name w, which is guaranteed to be fresh.

$$\text{(CloseL)}\ \frac{p \xrightarrow{\overline{x}\,(w)} p' \quad q \xrightarrow{x(w)} q'}{p|q \xrightarrow{\tau} (w)(p'|q')} \qquad \text{(CloseR)}\ \frac{p \xrightarrow{x(w)} p' \quad q \xrightarrow{\overline{x}\,(w)} q'}{p|q \xrightarrow{\tau} (w)(p'|q')}$$

The rules (CloseL) and (CloseR) transform the object of the communication over x in a private channel between p and q. Name extrusion is a convenient primitive for formalizing secure data transmission, as implemented e.g. via cryptographic protocols.

$$\text{(Rep)}\ \frac{p|!p \xrightarrow{\alpha} p'}{!p \xrightarrow{\alpha} p'}$$

The last rule deals with replication. It allows to replicate a process as many times as needed, in a reentrant fashion, without consuming it. Notice that $!p$ is able also to perform the synchronizations of $p|p$, if any. We conclude this section by showing an example of the use of the rule system.

Example 12.3 (A derivation)
Let us consider the following agent:

$$(((y)\overline{x}\ y.p) \mid q) \mid x(z).r$$

The process $(y)\overline{x}\ y.p$ would like to set up a private channel with $x(z).r$, which however should remain hidden to q.

By using the rule system:

$$\begin{array}{rrrl}
 & & (((y)\overline{x}\ y.p) \mid q) \mid x(z).r \xrightarrow{\alpha} q_1 & \nwarrow_{(Close),\ q_1=(w)(q_2|q_3),\ \alpha=\tau} \\
 & ((y)\overline{x}\ y.p) \mid q \xrightarrow{\overline{x}(w)} q_2 \qquad x(z).r \xrightarrow{x(w)} q_3 & & \nwarrow_{(ParL),\ q_2=q_4|q} \\
 & (y)\overline{x}\ y.p \xrightarrow{\overline{x}(w)} q_4 \qquad w \notin fn(q) \qquad x(z).r \xrightarrow{x(w)} q_3 & & \nwarrow_{(Open),\ q_4=q_5\{^w/_y\}} \\
\overline{x}\ y.p \xrightarrow{\overline{x}\ y} q_5 \qquad w \notin fn(q)\ \ w \notin fn((y).p) \qquad x(z).r \xrightarrow{x(w)} q_3 & & & \nwarrow_{(Out)+(In),\ q_3=r\{^w/_z\},\ q_5=p} \\
 & w \notin fn(q)\ \ w \notin fn((y).p)\ \ w \notin fn((z).r) & &
\end{array}$$

so we have:

$$\begin{aligned}
q_5 &= p \\
q_4 &= q_5\left\{^w/_y\right\} = p\left\{^w/_y\right\} \\
q_3 &= r\{^w/_z\} \\
q_2 &= q_4 \mid q = p\left\{^w/_y\right\} \mid q \\
q_1 &= (w)(q_2 \mid q_3) = (w)\left(p\left\{^w/_y\right\} \mid q\right) \mid (r\{^w/_z\})
\end{aligned}$$

In conclusion:

$$(((y)\overline{x}\ y.p) \mid q) \mid x(z).r \xrightarrow{\tau} (w)\left(p\left\{^w/_y\right\} \mid q\right) \mid (r\{^w/_z\})$$

under the conditions:

$$w \notin fn(q)\ \ w \notin fn((y).p)\ \ w \notin fn((z).r)$$

12.3. Structural Equivalence of π-calculus

As we have already noticed for CCS, there are different terms representing essentially the same process. As the complexity of the calculus increases, it is more and more convenient to manipulate terms up to some

intuitive structural axioms. In the following we denote by ≡ the least congruence over π-calculus processes that includes α-conversion of bound names and that is induced by the following set of axioms. The relation ≡ is called *structural equivalence*.

$$\begin{array}{rcl} p + \mathbf{nil} & \equiv & p \\ p \mid \mathbf{nil} & \equiv & p \\ (x)\,\mathbf{nil} & \equiv & \mathbf{nil} \\ [x = y]\,\mathbf{nil} & \equiv & \mathbf{nil} \end{array} \qquad \begin{array}{rcl} p + q & \equiv & q + p \\ p \mid q & \equiv & q \mid p \\ (y)(x)p & \equiv & (x)(y)p \\ [x = x]\,p & \equiv & p \end{array} \qquad \begin{array}{rcl} (p + q) + r & \equiv & p + (q + r) \\ (p \mid q) \mid r & \equiv & p \mid (q \mid r) \\ (x)(p \mid q) & \equiv & p \mid (x)q \text{ if } x \notin fn(p) \\ p \mid !p & \equiv & !p \end{array}$$

12.3.1. Reduction semantics

The operational semantics of π-calculus is much more complicated than that of CCS because it needs to handle name passing and scope extrusion. By exploiting structural equivalence we can define a so-called *reduction semantics* that is simpler to understand. The idea is to define an LTS with silent labels only that models all the interactions that can take place in a process, without considering interaction with the environment. This is accomplished by first rewriting the process to a structurally equivalent normal form and then by applying basic reduction rules. In fact it can be proved that for each π-calculus process p there exists:

- a finite number of names $x_1, x_2, ..., x_k$;
- a finite number of guarded sums $s_1, s_2, ..., s_n$;
- and a finite number of processes $p_1, p_2, ..., p_m$

such that

$$P \equiv (x_1)...(x_k)(s_1|...|s_n|!p_1|...!p_m)$$

Then, a reduction is either a silent action performed by some s_i or a communication from an input prefix of say s_i with an output prefix of say s_j. We write the reduction relation as a binary relation on processes using the notation $p \mapsto q$ for indicating that p reduces to q in one step. The rules defining the relation $\mapsto$ are the following:

$$\frac{}{\tau.p + s \mapsto p} \qquad \frac{}{(x(y).p_1 + s_1)|(\bar{x}z.p_2 + s_2) \mapsto p_1\left\{{}^{z}/_{y}\right\}|p_2}$$

$$\frac{p \mapsto p'}{p|q \mapsto p'|q} \qquad \frac{p \mapsto p'}{(x)p \mapsto (x)p'} \qquad \frac{p \equiv q \quad q \mapsto q' \quad q' \equiv p'}{p \mapsto p'}$$

The reduction semantics can be put in correspondence with the (silent transitions of the) labelled operational semantics by the following theorem.

Theorem 12.4 (Harmony Lemma)
For any π-calculus processes p, p' and any action α we have that:

1. $\exists q.\ p \equiv q \ \wedge\ q \xrightarrow{\alpha} p'$ *implies that* $\exists q'.\ p \xrightarrow{\alpha} q' \ \wedge\ q' \equiv p'$
2. $p \mapsto p'$ *iff* $\exists q.\ p \xrightarrow{\tau} q \ \wedge\ q \equiv p'$.

12.4. Abstract Semantics of π-calculus

Now we present an abstract semantics of π-calculus, namely we do not consider the internal structure of terms but focus on their behaviours. As we saw in CCS one of the main goals of abstract semantics is to find the correct degree of abstraction. Thus also in this case there are many kinds of bisimulations that lead to

different bisimilarities, which are useful in different circumstances depending on the properties that we want to study.

We start from *strong bisimulation* of π-calculus which is an extended version of the strong bisimulation of CCS. Then we will present the *weak bisimulation* for π-calculus. An important new feature of π-calculus is the choice of the time the names used as objects of input transitions are assigned their actual values. If they are assigned *before* the choice of the (bi)simulating transition, namely if the choice of the transition may depend on the assigned value, we get the *early* bisimulation. Instead, if the choice must hold for all possible names we have the *late* bisimulation case. As we will see in short, the latter option leads to a finer semantics.

12.4.1. Strong Early Ground Bisimulations

In *early* bisimulation we require that for each name w that an agent can receive on a channel x there exists a state q' in which the bisimilar agent will be after receiving w on x. This means that the bisimilar agent can choose a different transition (and thus a different state q') depending on the observed name w. Formally, a binary relation S on π-calculus agents is a *strong early ground bisimulation* if:

$$p\ S\ q \Rightarrow \begin{cases} \text{if } p \xrightarrow{\alpha} p' \text{ with } \alpha \neq x(y)\ \wedge\ bn(\alpha) \notin fn(q), \text{ then } \exists q'.\ q \xrightarrow{\alpha} q' \text{ and } p'\ S\ q' \\ \text{if } p \xrightarrow{x(y)} p' \text{ with } y \notin fn(q), \text{ then } \forall w.\ \exists q'.\ q \xrightarrow{x(y)} q' \text{ and } p'\{^w/_y\}\ S\ q'\{^w/_y\} \\ \text{(and vice versa)} \end{cases}$$

Two agents p and q are said to be *early bisimilar*, written $p \overset{\circ}{\sim}_E q$, iff:

$$p\ S\ q \text{ for some strong early ground bisimulation } S.$$

Notice that the conditions $bn(\alpha) \notin fn(q)$ and $y \notin fn(q)$ are required, since otherwise a bound name in the action which is fresh in p could be not fresh in q.

12.4.2. Strong Late Ground Bisimulations

In this case of late bisimulation, we require that, if an agent p can perform an input operation on a channel x, then there exists a state q' in which the bisimilar agent will be after receiving any possible value on x. Formally, a binary relation S on π-calculus agents is a *strong late ground bisimulation* if:

$$p\ S\ q \Rightarrow \begin{cases} \text{if } p \xrightarrow{\alpha} p' \text{ with } \alpha \neq x(y)\ \wedge\ bn(\alpha) \notin fn(q), \text{ then } \exists q'.\ q \xrightarrow{\alpha} q' \text{ and } p'\ S\ q' \\ \text{if } p \xrightarrow{x(y)} p' \text{ with } y \notin fn(q), \text{ then } \exists q'.\ q \xrightarrow{x(y)} q' \text{ and } \forall w.\ p'\{^w/_y\}\ S\ q'\{^w/_y\} \\ \text{(and vice versa)} \end{cases}$$

As usual we have that two agents p and q are said to be *late bisimilar*, written $p \overset{\circ}{\sim}_L q$ iff:

$$p\ S\ q \text{ for some strong late ground bisimulation } S.$$

Let us show an example which illustrates the difference between late and early bisimilarities.

Example 12.5 (Early vs late bisimulation)
Let us consider the processes:

$$\begin{aligned} p &= x(y).\tau.\,\mathbf{nil} + x(y).\,\mathbf{nil} \\ q &= p + x(y).[y = z].\tau.\,\mathbf{nil} \end{aligned}$$

The two processes p and q are early bisimilar. In fact, let q perform an input operation on x by choosing the right branch of the + operation. Then, if the received name y is equal to z, then p can choose to perform the left input operation and reach the state $\tau.\,\mathbf{nil}$ *which is equal to the state reached by q. Otherwise, if* $y \neq z$*, then the guard* $[y = z]$ *is not satisfied and q is blocked and p can choose to perform the right input and reach the state nil.*

On the contrary, if late bisimilarity is considered, then the two agents are not equivalent. In fact p should find a state which can handle all the possible value sent on x. If we choose to move on the left, the choice can work well when y = z but not in the other cases. On the other hand, if we choose to move on the right the choice does not work well with y = z.

The above example shows that late bisimulation is not coarser than early. In fact, it is possible to prove that late bisimulation is strictly finer than early.

12.4.3. Strong Full Bisimilarity

Unfortunately both early and late ground bisimilarities are not congruences, even in the strong case, as shown by the following counterexample.

Example 12.6 (Ground bisimilarities are not congruences)
Let us consider the following agents:

$$p = \overline{x}\,x.\,\mathbf{nil} \mid x'(y).\,\mathbf{nil} \qquad q = \overline{x}\,x.x'(y).\,\mathbf{nil} + x'(y).\overline{x}\,x.\,\mathbf{nil}$$

The agents p and q are bisimilar (according to both weak and strong bisimulation relations), as they generate isomorphic transition systems. Now, in order to show that ground bisimulations are not congruences, we define the following context:

$$C[\ \_\ ] = z(x')(\ \_\ )$$

by filling the hole of $C[\ \_\ ]$ once with p and once with q we obtain:

$$p' = C[p] = z(x')(\overline{x}\,x.\,\mathbf{nil} \mid x'(y).\,\mathbf{nil}) \qquad q' = C[q] = z(x')(\overline{x}\,x.x'(y).\,\mathbf{nil} + x'(y).\overline{x}\,x.\,\mathbf{nil})$$

p' and q' are not bisimilar. In fact, the agent p' can execute the input action $z(x)$ becoming the agent $\overline{x}\,x.\,\mathbf{nil} \mid x(y).\,\mathbf{nil}$ that can perform an internal synchronization τ; q' on the other hand cannot perform the same hidden action τ after executing the input action $z(x)$.

The problem illustrated by the previous example is due to aliasing, and it appears often in programming languages with both global variables and parameter passing to procedures. It can be solved by defining a finer relation between agents called *strong early full bisimilarity* and defined as follows:

$$p \sim_C q \Leftrightarrow p\sigma \stackrel{\circ}{\sim}_E q\sigma \ \text{ for every substitution } \sigma$$

where a substitution σ is a function from names to names that is equal to the identity function almost everywhere (i.e. it differs from the identity function only on a finite number of elements of the domain). It is possible to define *strong late full bisimilarity* in a similar way.

12.4.4. Weak Early and Late Ground Bisimulations

As for CCS, we can define the weak versions of bisimulation relations. The definition of weak early ground bisimulation is the following:

$$p\ S\ q \Rightarrow \begin{cases} \text{if } p \xrightarrow{\alpha} p' \text{ with } \alpha \neq x(y) \ \wedge\ bn(\alpha) \notin fn(q), \ \text{ then } \exists q'.\ q \stackrel{\alpha}{\Longrightarrow} q' \text{ and } p'\ S\ q' \\ \text{if } p \xrightarrow{x(y)} p' \text{ with } y \notin fn(q), \ \text{ then } \forall w.\ \exists q'.\ q \stackrel{x(y)}{\Longrightarrow} q' \text{ and } p'\{^w/_y\}\ S\ q'\{^w/_y\} \\ \text{(and vice versa)} \end{cases}$$

where here α could be τ. So we define the corresponding bisimilarity as follows:

$$p \stackrel{\bullet}{\approx}_E q \text{ iff } p\ S\ q \text{ for some weak early ground bisimulation } S.$$

The late version of the weak ground bisimulation is the following:

$$p\ S\ q \Rightarrow \begin{cases} \text{if } p \xrightarrow{\alpha} p' \text{ with } \alpha \neq x(y)\ \wedge\ bn(\alpha) \notin fn(q), \text{ then } \exists q'.\ q \overset{\alpha}{\Longrightarrow} q' \text{ and } p'\ S\ q' \\ \text{if } p \xrightarrow{x(y)} p' \text{ with } y \notin fn(q), \text{ then } \exists q'.\ q \overset{x(y)}{\Longrightarrow} q' \text{ and } \forall w.\ p'\{^w/_y\}\ S\ q'\{^w/_y\} \\ \text{(and vice versa)} \end{cases}$$

So we define the corresponding bisimilarity as follow:

$$p \overset{\bullet}{\approx}_L q \text{ iff } p\ S\ q \text{ for some weak late ground bisimulation } S.$$

As in the strong case, weak ground bisimilarities are not congruences due to aliasing. In addition, weak bisimilarities are not congruences for a + context, as it was already the case for CCS. Both problems can be fixed by combining the solutions we have shown for weak CCS and for π-calculus strong ground bisimilarities.

Part VI.

Probabilistic Models and PEPA

13. Measure Theory and Markov Chains

In this chapter we present models of concurrent systems which make use of probability theory. In these models probability and non-determinism are used and combined in order to model concurrency, parallelism and choices.

We have seen in the previous chapters how non-determinism allows us to represent choices and parallelism. Probability can be viewed as a refinement of non-determinism. We distinguish two main cases: *stochastic* and *probabilistic* models.

In *stochastic* models each event has a duration, which is defined in terms of a probability measure. The model binds a random variable to each operation, this variable represents the time necessary to execute the operation. So in this case probabilities allow one to order the operations over the time. Most models use exponentially distributed variables, associating a rate to each event. Often in stochastic systems also the non-deterministic choice is discarded, in such systems when a race between events happens the fastest operation is executed first.

Probabilistic models use probability to represent choices. These models associate a probability to each operation. If many operations are enabled at the same time, then the system uses the probability distribution to choose the operation which will be executed. As we will see many different combinations of probability and non-determinism have been studied.

We start this chapter by introducing some basics measure theory on which we will rely in order to construct probabilistic and stochastic models. Then we will present one of the most used stochastic models, called *Markov chains*. A Markov chain, named after the Russian mathematician Andrey Markov (1856–1922), is characterized by the the fact that the probability to evolve from one state to another depends only on the current state and not on the sequence of events that preceded it (e.g., it does not depend on the states traversed before reaching the current one). This specific kind of "memorylessness" is called the *Markov property*. A Markov chain allows to predict important statistical properties about system's future. We will discuss both the discrete time and the continuous time variants of Markov chains and we will see some interesting properties which can be studied relying on probability theory.

13.1. Measure Theory

13.1.1. σ-field

We start by introducing measure theory from the core concept of σ-field. A σ-field will be the starting point to define measurable spaces and hence probability spaces.

Definition 13.1 (σ-field)
Let Ω be a set and $\mathscr{A}$ be a family of subsets of Ω, then $\mathscr{A}$ is a σ-field iff:

1. $\varnothing \in \mathscr{A}$
2. $\forall A \in \mathscr{A} \Rightarrow (\Omega \setminus A) \in \mathscr{A}$
3. $\forall A_0, \ldots, A_n, \ldots \in \mathscr{A}$ *countable sequence of sets* $\Rightarrow \bigcup_{i \in \omega} A_i \in \mathscr{A}$

It is immediate to see that $\Omega \in \mathscr{A}$ (by 1 and 2) and that, due to 2, 3 and the De Morgan property also the intersection of a countable sequence (chain) of elements of $\mathscr{A}$ is in $\mathscr{A}$, i.e., $\bigcap_{i \in \omega} A_i = \Omega \setminus (\bigcup_{i \in \omega} (\Omega \setminus A_i))$.

A set contained in $\mathscr{A}$ is usually called a *measurable set*, and the pair $(\Omega, \mathscr{A})$ is called *measurable space*.

Let us illustrate the notion of σ-field by showing a simple example over a finite set of events.

Example 13.2
Let $\Omega = \{a, b, c, d\}$ be a set, we define a σ-field on Ω by setting $\mathscr{A} \subseteq 2^{\Omega}$:

$$\mathscr{A} = \{\varnothing, \{a, b\}, \{c, d\}, \{a, b, c, d\}\}$$

Measurable spaces set the domain on which we define a particular class of functions called *measures*, which assign a real number to each measurable set of the space.

Definition 13.3 (Measure on $(\Omega, \mathscr{A})$)
Let $(\Omega, \mathscr{A})$ be a measurable space, then a function $\mu : \mathscr{A} \to [-\infty, +\infty]$ is a measure iff:

- $\mu(\varnothing) = 0$
- $\forall A \in \mathscr{A}.\ \mu(A) \geq 0$
- $\forall A_1, \ldots, A_n, \ldots \in \mathscr{A}$ *countable sequence of pairwise disjoint sets* $\mu(\bigcup_{i\in\omega} A_i) = \sum_{i\in\omega} \mu(A_i)$

We are interested to a particular class of measures called *probabilities*. A probability is a *normalized* measure.

Definition 13.4 (Probability)
A measure P on $(\Omega, \mathscr{A})$ is called sub-probability *iff $P(\Omega) \leq 1$. Moreover if $P(\Omega) = 1$ then P is called a* probability *on $(\Omega, \mathscr{A})$.*

Definition 13.5 (Probability space)
Let Ω be a set, $\mathscr{A}$ be a σ-field on Ω and P be a probability measure on $(\Omega, \mathscr{A})$, then $(\Omega, \mathscr{A}, P)$ is called probability space.

13.1.2. Constructing a σ-field

Obviously one can think that in order to construct a σ-field that contains some sets equipped with a probability it is enough to construct the closure of these sets (together with top and bottom elements) under complement and countable union. But it came out from set theory that this is not possible if Ω is uncountable. In fact it has been shown that it is not possible to construct (in ZFC set theory) a measure on $2^{\aleph_0}$ (i.e. a function $P : 2^{\mathbb{R}} \longrightarrow [0, 1]$). So we have to find another way to define a σ-field on spaces that are uncountable.

We use an example which shows how this problem can be solved.

Example 13.6 (Coin tosses)
Let us consider the classic coin toss experiment. We have a fair coin and we want to model sequences of coin tosses. We would like to define Ω as follow:

$$\Omega = \{head, tail\}^{\infty}$$

Unfortunately this set has cardinality $2^{\aleph_0}$. As we have just said a measure on uncountable sets does not exist. So we restrict our attention on a countable set: the set τ of finite paths of coin tosses. In order to define a σ-field which allows to express almost all the events that we could express in words we define the following set for each $\alpha \in \tau$ called the shadow *of α:*

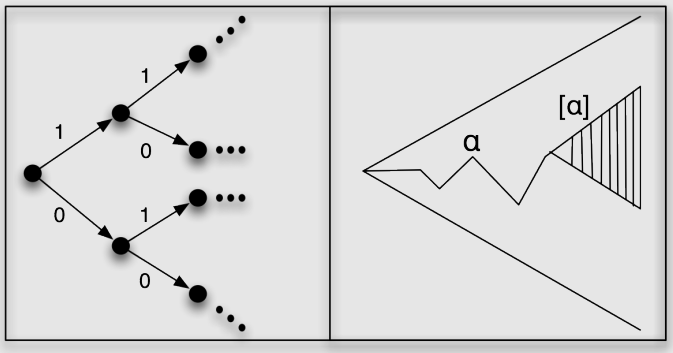
$$[\alpha] = \{\ \omega \mid \alpha\omega' = \omega,\ \textit{where } \omega \textit{ and } \omega' \textit{ are infinite paths}\ \}$$

Now the σ-field which we were looking for is that generated by the shadows of τ. In this way we can start by defining a probability measure P on the σ-field generated by the shadows of τ, then we can define the probability $p : \{head, tail\}^\infty \longrightarrow [0,1]$ of arbitrary sequences of coin tosses by setting:

$$p(\alpha) = \begin{cases} P([\alpha]) & \text{if } \alpha \text{ is finite} \\ P\Big(\bigcap_{\beta\in\tau,\ \alpha\in[\beta]} [\beta]\Big) & \text{if } \alpha \text{ is infinite} \end{cases}$$

For the second case, remind that the definition of σ-field ensures that countable intersection of measurable sets is measurable. Measure theory results show that this measure exists and is unique.

The right hand side of the picture shows graphically the set $[\alpha]$ of infinite paths corresponding to the finite path α.



Very often we have structures that are associated with a topology (e.g. there exists a standard topology associated to each CPO called Scott topology) so it is useful to define a standard method to obtain a σ-field from a topology.

Definition 13.7 (Topology)
Let X be a set and τ be a family of subsets of X. Then τ is said to be a topology *on X iff*

- $X, \emptyset \in \tau$
- $A, B \in \tau \Rightarrow A \cap B \in \tau$
- *let $\{A_i\}$ be any sequence of sets in τ then $\bigcup A_i \in \tau$*

We call A an open *set if it is in τ and* closed *set if $X \setminus A$ is open. The pair (X, τ) is said to be a* topological space.

Definition 13.8 (Borel σ-field)
Let T be a topology we will call Borel *σ-field of T the smallest σ-field that contains T.*

It turns out that the σ-field generated by the shadows which we have seen in the previous example is the Borel σ-field generated by the topology associated with the CPO of sets of infinite paths ordered by inclusion.

Definition 13.9 (Euclidean topology)
The euclidean topology *is a topology on real numbers whose open sets are open intervals of real numbers:*

$$]a, b[\quad = \quad \{x \in \mathbb{R} \mid a < x < b\}$$

We can extend the topology to the correspondent Borel σ-field, then associating to each open interval its length we obtain the Lebesgue *measure.*

13.1.3. Continuous Random Variables

As we said stochastic processes associate a *random variable* exponentially distributed to each event in order to represent its timing. So the concept of random variable and distribution will be central in next sections.

Definition 13.10 (Random variable)
Let $(\Omega, \mathscr{A}, P)$ be a probability space, a function $X : \Omega \to \mathbb{R}$ is said to be a random variable *iff*

$$\forall t \in \mathbb{R}.\ \{\omega \in \Omega \mid X(\omega) \leq t\} \in \mathscr{A}$$

Notice that if we take as $(\Omega, \mathscr{A})$ the measurable space of the real numbers with the Lebesgue measure, the identity $\mathbb{R} \to \mathbb{R}$ satisfies the above condition. As another example, we can take sequences of coin tosses and see them as binary representations of decimals in $[0, 1)$.

Random variables can be classified by considering the set of their values. We call *discrete* random variable a variable which has a numerable or finite set of possible values. We say that a random variable is *continuous* if the set of its values is continuous. Since we are particularly interested in continuous variables, in the remaining of this chapter we will consider only this type of variables. Note that almost all the definitions and theorem that we will see can be reformulated for the discrete case.

A random variable is completely characterized by its *probability law* which describes the probability that the variable will be found in a value less than or equal to the parameter.

Definition 13.11 (Cumulative distribution function)
Let $S = (\Omega, \mathscr{A}, P)$ be a probability space, $X : \Omega \to \mathbb{R}$ be a continuous random variable over S. We call cumulative distribution function *(*probability law*) of X the image of P through X and denote it by F_X:*

$$F_X(t) = P(\{\omega \in \Omega \mid X(\omega) \leq t\})$$

The other important function which describes the relative probability of a continuous random variable to take a specified value is the *probability density*.

Definition 13.12 (Probability density)
Let $S = (\Omega, \mathscr{A}, P)$ be a probability space, $X : \Omega \to \mathbb{R}$ be a continuous random variable over S, we call f the probability density *of X iff:*

$$\forall a, b \in \mathbb{R}.\ P(\{\omega \in \Omega \mid a \leq X(\omega) \leq b\}) = \int_a^b f(x)dx$$

So we can define the law F_X of a variable X with density f as follows:

$$F_X(t) = \int_{-\infty}^{t} f(x)dx$$

From now on we will write $P(X = a)$ by meaning $P(\{\omega \mid X(\omega) = a\})$ and we will write $P(a \leq X \leq b)$ by meaning $P(\{\omega \in \Omega \mid a \leq X(\omega) \leq b\})$.

As we said we are particularly interested in exponentially distributed random variables.

Definition 13.13 (Exponential distribution)
A continuous random variable X is said to be exponentially distributed *with parameter λ if its probability law and density function are defined as follows:*

$$F_X(x) = \begin{cases} 1 - e^{-\lambda x} & \text{if } x \geqslant 0 \\ 0 & x < 0 \end{cases} \qquad f(x) = \begin{cases} \lambda e^{-\lambda x} & \text{if } x \geqslant 0 \\ 0 & x < 0 \end{cases}$$

The parameter λ is called the rate *of X.*

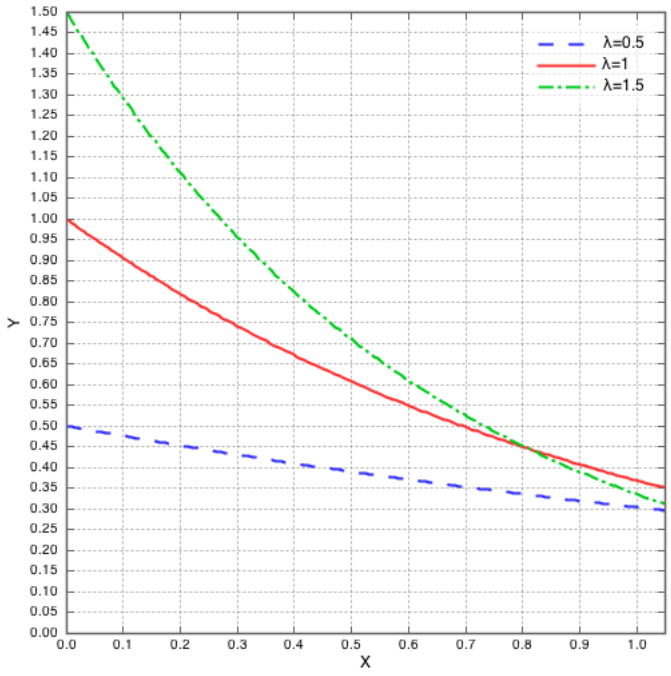


Figure 13.1.: Exponential density distributions

Random variables can be used to represent the timing of events: from now on we will interpret a variable as a function which given an event (i.e., an element of Ω) returns the time at which it is executed.

One of the most important features of exponentially distributed random variables is that they are memoryless, meaning that the current value of the random variable does not depend on the previous values. Let us show this concept with a simple example.

Example 13.14 (Radioactive Atom)
Let us consider a radioactive atom, which due to its instability can easily loose energy. It turns out that the probability that an atom will decay is constant over the time. So this system can be modelled by using an exponentially distributed continuous random variable whose rate is the decay rate of the atom. Since the random variable is memoryless we have that the probability that the atom will decay at time $t_0 + t$ knowing that it is not decaying yet at time t_0 is the same for any choice of t_0.

Theorem 13.15 (Memoryless)
Let d be an exponentially distributed continuous random variable with rate λ then we have:

$$P(d \leq t_0 + t \mid d > t_0) = P(d \leq t)$$

Proof. Since d is exponentially distributed we have that its cumulative probability is defined as follows:

$$F_d(t) = \int_0^t \lambda e^{-\lambda x} dx$$

so we want to show:

$$\frac{\int_{t_0}^{t_0+t} \lambda e^{-\lambda x} dx}{\int_{t_0}^{\infty} \lambda e^{-\lambda x} dx} \stackrel{?}{=} \int_0^t \lambda e^{-\lambda x} dx$$

Since $\int_a^b \lambda e^{-\lambda x} dx = \left[-e^{-\lambda x}\right]_a^b = \left[e^{-\lambda x}\right]_b^a$ thus:

$$\frac{\int_{t_0}^{t_0+t} \lambda e^{-\lambda x} dx}{\int_{t_0}^{\infty} \lambda e^{-\lambda x} dx} = \frac{\left[e^{-\lambda x}\right]_{t_0+t}^{t_0}}{\left[e^{-\lambda x}\right]_{\infty}^{t_0}} = \frac{e^{-\lambda t_0} - e^{-\lambda t} \cdot e^{-\lambda t_0}}{e^{-\lambda t_0}} = \frac{\cancel{e^{-\lambda t_0}}(1 - e^{-\lambda t})}{\cancel{e^{-\lambda t_0}}} = 1 - e^{-\lambda t}$$

We conclude by:

$$\int_0^t \lambda e^{-\lambda x} dx = \left[e^{-\lambda x}\right]_t^0 = 1 - e^{-\lambda t}$$

□

Another interesting feature of exponentially distributed random variables is the easy way in which we can compose information in order to find the probability of more complex events. For example if we have two random variables d_1 and d_2 which represent the delay of two events e_1 and e_2, we can try to calculate the probability that either of the two events will be executed before a specified time t. As we will see it happens that we can define an exponentially distributed random variable whose cumulative probability is the probability that either e_1 or e_2 executes before a specified time t.

Theorem 13.16
Let d_1 and d_2 be two exponentially distributed continuous random variables with rate respectively λ_1 and λ_2 then:

$$P(\min\{d_1, d_2\} \leq t) = 1 - e^{-(\lambda_1+\lambda_2)t}$$

Proof. Easily we have:

$$\begin{aligned} P(\min\{d_1, d_2\} \leq t) &= P(d_1 \leq t) + P(d_2 \leq t) - P(d_1 \leq t \wedge d_2 \leq t) \\ &= (1 - e^{-\lambda_1 t}) + (1 - e^{-\lambda_2 t}) - (1 - e^{-\lambda_1 t})(1 - e^{-\lambda_2 t}) \\ &= 1 - e^{-\lambda_1 t} \cdot e^{-\lambda_2 t} \\ &= 1 - e^{-(\lambda_1+\lambda_2)t} \end{aligned}$$

□

A second important value that we can calculate is the probability that an event will be executed before another. This corresponds in our view to calculate the probability that d_1 will take a value smaller than that of d_2.

Theorem 13.17
Let d_1 and d_2 be two exponentially distributed continuous random variables with rate respectively λ_1 and λ_2 then:

$$P(d_1 < d_2) = \frac{\lambda_1}{\lambda_1 + \lambda_2}$$

Proof. Easily we have:

$$\begin{aligned}
\int_0^\infty \lambda_1 e^{-\lambda_1 t_1} \left(\int_{t_1}^\infty \lambda_2 e^{-\lambda_2 t_2} dt_2 \right) dt_1 &= \int_0^\infty \lambda_1 e^{-\lambda_1 t_1} \left[e^{-\lambda_2 t_2} \right]_\infty^{t_1} dt_1 \\
&= \int_0^\infty \lambda_1 e^{-\lambda_1 t_1} \cdot e^{-\lambda_2 t_1} dt_1 \\
&= \int_0^\infty \lambda_1 e^{-(\lambda_1 + \lambda_2) t_1} dt_1 \\
&= \left[\frac{\lambda_1}{\lambda_1 + \lambda_2} e^{-(\lambda_1 + \lambda_2) t} \right]_\infty^0 \\
&= \frac{\lambda_1}{\lambda_1 + \lambda_2}
\end{aligned}$$

□

13.2. Stochastic Processes

Stochastic processes are a very powerful mathematical tool that allows us to describe and analyse a wide variety of systems.

Definition 13.18 (Stochastic process)
Let $(\Omega, \mathscr{A}, P)$ be a probability space and T be a set, then a family $\{X_t\}_{t \in T}$ of random variables over Ω is said to be a stochastic process. *A stochastic process can be identified with a function $X : \Omega \times T \longrightarrow \mathbb{R}$ such that:*

$$\forall t \in T.\ X(\_, t) : \Omega \longrightarrow \mathbb{R} \text{ is a random variable}$$

Usually the values in Ω that each random variable can take are called states *and the element of T are interpreted as times.*

Obviously the set T strongly characterizes the process. A process in which T is $\mathbb{N}$ or a subset of $\mathbb{N}$ is said to be a *discrete time* process, on the other hand if $T = \mathbb{R}$ then the process is a *continuous time* process. The same distinction is usually done on the value that each random variable can assume: if this set has a countable or finite cardinality then the process is *discrete*; otherwise it is *continuous*. We will focus only on discrete processes with both discrete and continuous time.

13.3. Markov Chains

Stochastic processes studied by classical probability theory often involve only independent variables, namely the outcomes of the process are totally independent from the past. *Markov chains* extend the classic theory by dealing with processes where each variable is influenced by the previous one. This means that in Markov

processes the next outcome of the system is influenced only by the previous state. One could think to extend this theory in order to allow general dependencies between variables, but it turns out that it is very difficult to prove general results on processes with dependent variables. We are interested in Markov chains since they provide a mathematical framework to represent and analyse interleaving and sequential systems.

Definition 13.19 (Markov chain)
Let $(\Omega, \mathscr{A}, P)$ be a probability space, T be a set and $\{X_t\}_{t\in T}$ be a stochastic process. Then, $\{X_t\}_{t\in T}$ is said to be a Markov chain *if for each sequence $t_{n+1} > t_n > \ldots > t_0$ of times in T and for any measurable subset of states A (i.e., $A \in \mathscr{A}$) we have:*

$$P(X_{t_{n+1}} \in A \mid X_{t_n} = x_n, \ldots, X_{t_0} = x_0) = P(X_{t_{n+1}} \in A \mid X_{t_n} = x_n)$$

The previous proposition is usually referred to as Markov property.

An important characteristic of a Markov chain is the way in which it is influenced by the time. We have two type of Markov chains, *inhomogeneous* and *homogeneous*. In the first case the state of the system depends on the time, namely the probability distribution changes over the time. In homogeneous chains on the other hand the time does not influence the distribution, i.e., the transition probability does not change during the time. We will consider only the simpler case of homogeneous Markov chains gaining the possibility to shift the time axis back and forward.

Definition 13.20 (Homogeneous Markov chain)
Let $\{X_t\}_{t\in T}$ be a Markov chain then it is said to be homogeneous if for any measurable set A and for each $t', t \in T$ with $t' > t$ we have:

$$P(X_{t'} \in A | X_t = x) = P(X_{t'-t} \in A | X_0 = x)$$

13.3.1. Discrete and Continuous Time Markov Chain

As we said one of the most important thing about stochastic processes in general, and about Markov chains in particular, is the choice of the set of times. In this section we will introduce two kinds of Markov chain, those in which $T = \mathbb{N}$ called *discrete time Markov chain* (DTMC) and those in which $T = \mathbb{R}$ referred as *continuous time Markov chain*.

Definition 13.21 (Discrete time Markov Chain (DTMC))
Let $\{X_t\}_{t\in\mathbb{N}}$ be a stochastic process then it is a discrete time Markov chain *(DTMC) iff:*

$$P(X_{n+1} = x_{n+1} \mid X_n = x_n, \ldots, X_0 = x_0) = P(X_{n+1} = x_{n+1} \mid X_n = x_n) \text{ for } n \in \mathbb{N}$$

Since we are restricting our attention to homogeneous chains then we can reformulate the Markov property as follows:

$$P(X_{n+1} = x_{n+1} \mid X_n = x_n, \ldots, X_0 = x_0) = P(X_1 = x_{n+1} \mid X_0 = x_n)$$

Definition 13.22 (Continuous time Markov Chain (CTMC))
Let $\{X_t\}_{t\in\mathbb{R}}$ be a stochastic process then it is a continuous time Markov chain *(CTMC) if for any $\Delta_t \in \mathbb{R}$ and any sequence $t_n + \Delta_t > t_n > \ldots > t_0$ we have :*

$$P(X_{t_n+\Delta_t} = x \mid X_{t_n} = x_n, \ldots, X_{t_0} = x_0) = P(X_{t_n+\Delta_t} = x \mid X_{t_n} = x_n)$$

As for the discrete case the homogeneity allows to reformulate the Markov property as follows:

$$P(X_{t_n+\Delta_t} = x \mid X_{t_n} = x_n, \ldots, X_{t_0} = x_0) = P(X_{\Delta_t} = x \mid X_0 = x_n)$$

So from now on we use the term "Markov chain" as a synonym for "homogeneous Markov chain". We remark that the exponential random variable is the only continuous random variable with the memoryless property, i.e., CTMC are exponentially distributed.

13.3.2. DTMC as LTS

A DTMC can be viewed as a particular LTS whose labels are probabilities. Usually such LTS are called *probabilistic transition systems* (PTS). The PTS (S, α) associated with a DTMC has a transition function of the type $\alpha : S \longrightarrow (D(S) + 1)$ where $D(S)$ is the set of discrete probability distributions over S and $1 = \{*\}$ is a singleton representing the deadlock states. So if $\{X_t\}_{t\in\omega}$ is a DTMC whose set of states is S we can define a PTS by taking S as the set of states of the transition system and defining the transition function $\alpha : S \longrightarrow (D(S) + 1)$ as follows:

$$\alpha(s) = \begin{cases} \lambda s'.P(X_1 = s' \mid X_0 = s) & \text{if } s \text{ is not a deadlock state} \\ * & \text{otherwise} \end{cases}$$

Note that for each state s, which is not a deadlock state, it holds:

$$\sum_i \alpha(s)(s_i) = 1$$

A difference between LTS and PTS is that in LTS we can have structures like that shown in Figure 13.2 (a). In PTS we cannot have this kind of situation since two different transitions between the same pair of states have the same meaning of a single transition labeled with the sum of the probabilities, as shown in Figure 13.2 (b).

(a) (b)

Figure 13.2.: Two equivalent DTMCs

Usually the transition function is represented through a matrix P whose indices i, j represent states s_i, s_j and each element $a_{i,j}$ is the probability that knowing that the system is in the state i it would be in the state j in the next time instant, namely $\forall i, j \in |\ S\ | \ .\ a_{i,j} = \alpha(s_i)(s_j)$, note that in this case each row of P must sum to one. This representation allows us to study the system by relaying on linear algebra. In fact we can represent the present state of the system by using a row vector $\Pi^{(t)} = [\pi_i^{(t)}]_{i\in S}$ where $\pi_i^{(t)}$ represents the probability that the system is in the state i at the time t. If we want to calculate how the system will evolve (i.e., the next state distribution) starting from this state we can simply multiply the vector with the matrix which represents the transition function as follows:

$$\Pi^{(t+1)} = \Pi^t * P = \begin{vmatrix}\pi_1 & \pi_2 & \pi_3\end{vmatrix} \begin{vmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{vmatrix} = \begin{vmatrix} a_{1,1} * \pi_1 + a_{2,1} * \pi_2 + a_{3,1} * \pi_3 \\ a_{1,2} * \pi_1 + a_{2,2} * \pi_2 + a_{3,2} * \pi_3 \\ a_{1,3} * \pi_1 + a_{2,3} * \pi_2 + a_{3,3} * \pi_3 \end{vmatrix}^T$$

As we will see for some special class of DTMCs we can prove the existence of a limit vector for $t \longrightarrow \infty$, that is to say the probability that the system is found in a particular state is stationary in the long run.

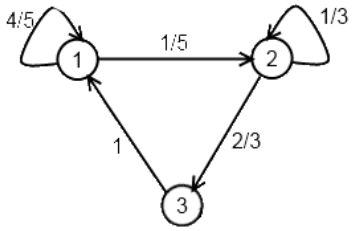


Figure 13.3.: DTMC

Example 13.23 (DTMC)
Let us consider the DTMC in Figure 13.3. We represent the chain algebraically by using the following matrix:

$$P = \begin{vmatrix} 4/5 & 1/5 & 0 \\ 0 & 1/3 & 2/3 \\ 1 & 0 & 0 \end{vmatrix}$$

Now suppose that we do not know the state of the system at time t, thus we assume the system has probability $\frac{1}{3}$ *of being in one of the three states. We represent this situation with the following vector:*

$$\Pi^{(t)} = \begin{vmatrix} 1/3 & 1/3 & 1/3 \end{vmatrix}$$

Now we can calculate the state distribution at time $t + 1$ *as follows:*

$$\begin{vmatrix} 1/3 & 1/3 & 1/3 \end{vmatrix} \begin{vmatrix} 4/5 & 1/5 & 0 \\ 0 & 1/3 & 2/3 \\ 1 & 0 & 0 \end{vmatrix} = \begin{vmatrix} 3/5 & 8/45 & 2/9 \end{vmatrix}$$

Notice that the sum of probabilities in the new state is again 1. Obviously we can iterate this process in order to simulate the evolution of the system.

Since we have represented a Markov chain by using a transition system it is quite natural to ask for the probability of a finite path.

Definition 13.24 (Finite path probability)
Let $\{X_t\}_{t\in\omega}$ *a DTMC and* $s_1, \ldots, s_n$ *a finite path of its PTS (i.e.* $\forall 1 \leq i < n.\ \alpha(s_i)(s_{i+1}) > 0$*) we define the probability of the path as follows:*

$$P(s_1\ \ldots\ s_n) = \Pi_{i=1}^{n-1} \alpha(s_i)(s_{i+1})$$

Example 13.25 (Finite paths)
Let us consider the DTMC of Example 13.23 and take the path 1 2 3 1*. We have:*

$$P(1\ 2\ 3\ 1) = \frac{1}{5} \cdot \frac{2}{3} \cdot 1 = \frac{2}{15}$$

Note that if we consider the sequence of states 1 1 3 1*:*

$$P(1\ 1\ 3\ 1) = \frac{4}{5} \cdot 0 \cdot 1 = 0$$

In fact there is no transition allowed from state 1 *to* 3*.*

Note that it would make little sense to define the probability of infinite paths as the product of each choice, in this case each infinite sequence would have a null probability. Obviously as we said we can avoid this problem by using the Borel σ-field generated by the shadows as we saw in the example 13.6.

Example 13.26 (Random walk with barrier)
Let us consider a DTMC with a countable set of states. Each state bigger than 0 *has two outcome transitions one which lead to the next state and one which lead to the previous one. The state* 0 *has only one arrow to the state* 1*. The structure of each state is described as follows:*

$$m \xrightarrow{1/2} m-1$$
$$m \xrightarrow{1/2} m+1$$

At the beginning we impose $P(0) = 1$*, namely the system starts at* 0*. Now it could be shown that the probability that an infinite path hits* 0 *after some time is* 1*.*

13.3.3. DTMC Steady State Distribution

In this section we will present a special class of DTMCs which guarantee that the probability that the system is found in a state does not change on the long term. This means that the distribution of each state of the DTMC (i.e. the correspondent value in the vector $\Pi^{(t)} = \left|\pi_1^{(t)} \dots \pi_n^{(t)}\right|$) reach a *steady state distribution* which does not change in the future, namely if π_i is the steady state distribution for the state i if $\pi_i^{(0)} = \pi_i$ then $\pi_i^{(t)} = \pi_i$ for each $t > 0$.

Definition 13.27 (Steady state distribution)
We define the steady state distribution $\Pi = \left|\pi_1 \dots \pi_n\right|$ *of a DTMC as:*

$$\pi_i = \lim_{t \longrightarrow \infty} \pi_i^{(t)}$$

when such limit exists.

In order to guarantee that the limit exists we will restrict our attention to a subclass of Markov chains.

Definition 13.28 (Ergodic Markov chain)
Let $\{X_t\}_{t \in T}$ *be a Markov chain then it is said to be* ergodic *if it has the following properties:*

i) irreducibility:each state is reachable from each other.

ii) aperiodicity: the MCD of the lengths of all paths from any state to itself must be 1.

Theorem 13.29
Let $\{X_t\}_{t\in T}$ be an ergodic homogeneous Markov chain then the steady state probability Π always exists and it is independent from the initial state probability distribution.

Probability distribution Π can be computed by solving the following system of linear equations: $\Pi = \Pi * P$ where P is the matrix associated to the chain, under the constraint that the sum of all probabilities be 1. It is possible to prove that the solution of such a system always exists and it is unique.

Example 13.30 (Steady state distribution)
Let us consider the DTMC of the example13.23. Let us find the steady state distribution. We have to solve the following linear system:

$$\begin{vmatrix}\pi_1 & \pi_2 & \pi_3\end{vmatrix} \begin{vmatrix}4/5 & 1/5 & 0\\ 0 & 1/3 & 2/3\\ 1 & 0 & 0\end{vmatrix} = \begin{vmatrix}\pi_1 & \pi_2 & \pi_3\end{vmatrix}$$

By solving the system of linear equations we obtain the solution:

$$\begin{vmatrix}10/3 * \pi_2 & \pi_2 & 2/3 * \pi_2\end{vmatrix}$$

i.e., $\pi_1 = \frac{10}{3}\pi_2$ and $\pi_3 = \frac{2}{3}\pi_2$.
Now by imposing $\pi_1 + \pi_2 + \pi_3 = 1$ we have $\pi_2 = 1/5$ thus:

$$\Pi = \begin{vmatrix}2/3 & 1/5 & 2/15\end{vmatrix}$$

So it is more likely to find the system in the state 1 than in states 2 and 3 in the long run, because the steady state probability of this state is larger than the probabilities of the other two states.

13.3.4. CTMC as LTS

Also in the continuous case Markov chains can be represented as LTSs (S, α). Labels of the LTS which represents a CTMC will be rates. We have two equivalent definitions for the transition function:

$$\alpha : S \to S \to \mathbb{R}$$

$$\alpha : (S \times S) \to \mathbb{R}$$

Where S is the set of states of the chain and $\mathbb{R}$ would represent the rate which will label the transition. Also in this case we have that two different transitions between two states would be represented by only one transition summing the rates as was for the LTS associated to a DTMC. Moreover we have that the self loops can be ignored, this is due to the fact that in continuous time we allow the system to *sojourn* in a state for a period. As for DTMCs we can represent a CTMC by using linear algebra. In this case the matrix Q which represents the system is defined by setting $q_{i,j} = \alpha(i)(j)$, this matrix is usually called *infinitesimal generator*.

The probability that no transition happens from a state i in some time r is 1 minus the probability that some transition happens, which has a rate that is the sum of the rates of all the transitions outgoing from i:

$$\forall r \in (0, \infty).\ P(X(t + r) = i \mid X(t) = i) = e^{-\lambda r} \text{ with } \lambda = \sum_{j \neq i} q_{i,j}$$

where X is the random variable describing the state of the process at a given time ($X(t) = i$ means that the process is in a state i at time t).

13.3.5. Embedded DTMC of a CTMC

Often the study of a CTMC results very hard particularly in term of computational complexity. So it is useful to have a standard way to discretize the CTMC obtaining the *embedded* DTMC in order to simplify the analysis. Let C be a CTMC whose transition function is α_C then we define the embedded DTMC D of C as follows. The set of states of D is the same as C and each element of the one-step transition probability matrix of the Embedded DTMC represents the conditional probability of transitioning from state s_i into state s_j:

$$\alpha_D(s_i)(s_j) = \begin{cases} \frac{\alpha_c(s_i)(s_j)}{\sum_{s \neq s_i} \alpha_c(s_i)(s)} & \text{if } s_i \neq s_j \\ 0 & \text{otherwise} \end{cases}$$

as we can see the previous definition simply normalizes to 1 the rates in order to calculate a probability.

While the Embedded DTMC completely determines the probabilistic behaviour of the embedded discrete-time Markov chain, it does not fully capture the behaviour of the continuous-time process because it does not specify the rates at which transitions occur.

Regarding the steady state analysis, it is easy to notice that, as in the infinitesimal generator matrix Q describing the CTMC we have $q_{i,i} = -\sum_{j \neq i} q_{i,j}$, the steady state distribution can be computed by solving the equation $\pi \cdot Q = 0$.

13.3.6. CTMC Bisimilarity

Obviously, since Markov chains can be seen as a particular type of LTS one could think to modify the notion of bisimilarity in order to study the equivalence between stochastic systems.

Let us start by defining the notion of LTS bisimilarity in a slightly different way from that seen in Chapter 10. We define a function $\gamma : S \times Act \times 2^S \longrightarrow \{true, false\}$ which takes a state p, an action μ and a set of states I and returns true if there exists a state q in I reachable from p with a transition labelled by μ, formally:

$$\gamma(p, \mu, I) = \exists q \in I.\ p \xrightarrow{\mu} q$$

Now we define the bisimilarity on a LTS as follows:

$$p\ \phi(R)\ q \ \Leftrightarrow\ (\ \forall \mu \in Act, I \in R.\ \gamma(p, \mu, I) \Leftrightarrow \gamma(q, \mu, I)\ )$$

This means that two states are said to be bisimilar if they have the same function γ whatever way we choose an equivalence class I of R and an action μ.

Now we extend this construction to a CTMC. We define a function $\gamma_M : S \times 2^S \to \mathbb{R}$ simply by extending the transition function to sets of states as follows:

$$\gamma_M(s, I) = \sum_{s' \in I} \alpha(s)(s')$$

From now on we will use α also for its extension on sets of states $\alpha(s)(I)$. As we have just done for LTS we define the bisimilarity on CTMCs as follows:

$$s_1\ \varphi\ (R) s_2 \Longrightarrow \forall I \in R.\ \gamma_M(s_1, I) = \gamma_M(s_2, I)$$

$$\simeq = \bigsqcap_{R = \varphi(R)} R \qquad \text{CTMC bisimilarity}$$

meaning that the total rate of reaching from s_1 and s_2 any equivalence class of R is the same

Let us show how this construction works with an example.

Example 13.31
Let us consider the two CTMCs in Figure 13.4. We argue that the following relation identifies the classes of bisimilar states:

$$R = \{\ \{a_1, a_2\}, \{b_1, b_2, b_3\}, \{c_1, c_2\}, \{d_1, d_2\}\ \}$$

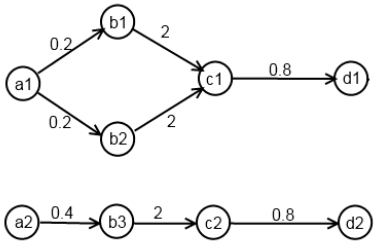


Figure 13.4.: CTMC bisimilarity

Let us show that R is a bisimulation:

$$\gamma_M(a_1, \{b_1, b_2, b_3\}) = \gamma_M(a_2, \{b_1, b_2, b_3\}) = 0.4$$

$$\gamma_M(b_1, \{c_1, c_2\}) = \gamma_M(b_2, \{c_1, c_2\}) = \gamma_M(b_3, \{c_1, c_2\}) = 2$$

$$\gamma_M(c_1, \{d_1, d_2\}) = \gamma_M(c_2, \{d_1, d_2\}) = 0.8$$

note that we have not mentioned all remaining trivial cases, where γ_M returns 0.

13.3.7. DTMC Bisimilarity

One could think that the same argument that we have just used for CTMCs can be also extended to DTMCs. It is easy to show that if a DTMC has no deadlock states, in particular if it is ergodic, then the bisimilarity become trivial. In fact we have $\gamma_M(s, S) = \sum_{s' \in S} \alpha(s)(s') = 1$ for each state s, i.e., all states are bisimilar. This does not mean that the concept of bisimulation on ergodic DTMCs is useless, in fact these relations can be used to factorize the chain (lumping) in order to study particular properties.

On the other hand if we consider DTMCs with some deadlock states the bisimilarity is not the trivial equivalence relation. We define $\gamma_M : S \longrightarrow 2^S \rightarrow (\mathbb{R} + 1)$ as follows:

$$\gamma_M(s)(I) = \begin{cases} * & \text{if } \alpha(s) = * \\ \sum_{s' \in I} \alpha(s)(s') & \text{otherwise} \end{cases}$$

In this case we have that the bisimilarity will have an equivalence class containing exactly all deadlock states. In fact, recalling the definition of bisimulation R, we have:

$$s_1 \; \varphi \; (R) s_2 \Longrightarrow \forall I \in R.\; \gamma_M(s_1)(I) = \gamma_M(s_2)(I)$$

Therefore, any two deadlock states s_1, s_2 are bisimilar ($\forall I \in 2^S.\; \gamma_M(s_1)(I) = \gamma_M(s_2)(I) = *$) and separated from any non deadlock state s ($\forall I \in 2^S.\; \gamma_M(s_1)(I) = * \neq \gamma_M(s)(I) \in \mathbb{R}$).

14. Markov Chains with Actions and Non-determinism

In this chapter we introduce some *probabilistic models* which can be defined by modifying the transition function of PTSs. As we have seen for Markov chains, the transition system representation is very useful since it comes with a notion of bisimilarity. In fact, using the advanced, categorical notion of *coalgebra* there is a standard method to define bisimilarity just according to the type of the transition function. Also a corresponding notion of Hennessy-Milner logic can be defined accordingly.

First we will see two different ways to add actions to our probabilistic models, then we will present extensions which combine non-determinism, actions and probabilities.

14.1. Discrete Markov Chains With Actions

In this section we show how it is possible to change the transition function of the LTS in order to extend Markov chains with labels that represent actions performed by the system. There are two main cases to consider, called *reactive models* and *generative models*, respectively. In the first case we add actions that are used by the controller to stimulate the system. When we want the system to change its state we give an input action to it which could determine its future state (its reaction). This is the reason why this type of models is called "reactive". In the second case the actions would represent the outcomes of the system, this means that when the system changes its state it shows an action, whence the terminology "generative". Formally we have:

$\alpha_R : S \to L \to (D(S) + 1)$ reactive probabilistic transition system (also called Markov decision processes)
$\alpha_G : S \to (D(L \times S) + 1)$ generative probabilistic transition system

where we recall that $D(S)$ is the set of discrete probability distributions over S.

We have that in a reactive system for each $s \in S$ and for every $l \in L$:

$$\sum_{s' \in S} \alpha_R \, s \, l \, s' = 1$$

On the other hand in a generative system for each $s \in S$:

$$\sum_{(l,s') \in L \times S} \alpha_G \, s \, (l, s') = 1$$

This means that in reactive systems the probability of every action must sum to 1 (w.r.t. a given state), while in a generative system the distribution of every state must sum to 1.

14.1.1. Reactive DTMC

Let us illustrate how a reactive system works by using a simple example.

Example 14.1 ("Random" coffee maker)
Let us consider a system which we call "random" coffee maker, in which the user can insert a coin (1 or 2 euros) then, the coffee maker, based on the value of the input, chooses to make a coffee or a cappuccino with larger or smaller probabilities. The system is represented in Figure 14.1. Note that, since we want to allow the system to take input from the environment we have chosen a reactive system to represent the

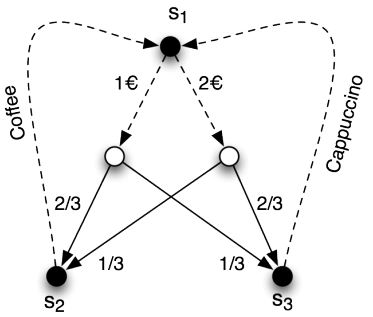


Figure 14.1.: A reactive PTS which represents a coffee maker

coffee maker. The set of labels is $L = \{1€, 2€, \textbf{Coffee}, \textbf{Cappuccino}\}$ *and the corresponding transitions are represented as dashed arrows. There are three states* s_1*,* s_2 *and* s_3*, represented with black-filled circles. If the input* $1€$ *is received in state* s_1*, then we can reach state* s_2 *with probability* $\frac{2}{3}$ *or* s_3 *with probability* $\frac{1}{3}$*, as illustrated by the solid arrows departing from the white-filled circle associated with the distribution. Vice versa, if the input* $2€$ *is received in state* s_1*, then we can reach state* s_2 *with probability* $\frac{1}{3}$ *or* s_3 *with probability* $\frac{2}{3}$*. From state* s_2 *there is only one transition available, with label* **Coffee***, that leads to* s_1 *with probability* 1*. Here the white-filled circle is omitted because the probability distribution is trivial. Similarly, from state* s_3 *there is only one transition available, with label* **Cappucino***, that leads to* s_1 *with probability* 1*.*

As we said using LTS we have a standard method to define bisimilarity between probabilistic systems. A relation R between states of reactive systems is a bisimulation if for every equivalence class I of R and for each action l we have:

$$s_1 \; R \; s_2 \Longrightarrow \alpha(s_1)\,(l)\,(I) = \alpha(s_2)\,(l)\,(I)$$

Note that two states s_1 and s_2 in order to be related must have for each action the same probability to reach the states in the equivalence class. Two states are said to be bisimilar if there exists a bisimulation in which they are related.

14.1.1.1. Larsen-Skou Logic

Now we will present a probabilistic version of Hennessy-Milner logic. This logic has been introduced by Larsen and Skou, and provides a new version of the modal operator. As usual we start from the syntax of the Larsen-Skou logic:

$$\varphi ::= \textbf{true} \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \langle l \rangle_q \, \varphi$$

The novelty resides in the new modal operator $\langle l \rangle_q \, \varphi$ that takes three parameters, a formula φ, an action l and a real number $q \leq 1$. Informally, the formula $\langle l \rangle_q \, \varphi$ expresses the possibility to reach a state satisfying the formula φ by performing the action l with probability at least q.

As we have done for Hennessy-Milner logic we present the Larsen-Skou logic by defining a satisfaction relation:

$$\begin{array}{lcl} s \models \textbf{true} & & \\ s \models \varphi_1 \wedge \varphi_2 & \Leftrightarrow & s \models \varphi_1 \text{ and } s \models \varphi_2 \\ s \models \neg\varphi & \Leftrightarrow & \neg s \models \varphi \\ s \models \langle l \rangle_q \, \varphi & \Leftrightarrow & \alpha \, s \, l \, [\![\varphi]\!] \geq q \text{ where } [\![\varphi]\!] = \{s \in S \mid s \models \varphi\} \end{array}$$

A state s verifies the formula $s \models \langle l \rangle_q \, \varphi$ if the probability to pass in a state that verifies φ from s with an action labelled l is bigger than or equal to q. Note that the corresponding modal operator of the Hennessy-Milner logic can be obtained by setting $q = 1$.

As it was the case for Hennessy-Milner logic, also in this case logic formulas characterize the bisimilarity. Moreover we have an additional strong result, in fact it can be shown that it is enough to consider only the version of the logic without negation.

Theorem 14.2 (Larsen-Skou bisimilarity characterization)
Two states of a reactive transition system are bisimilar iff they satisfy the same formulas of Larsen-Skou logic without negation.

Example 14.3 (Larsen-Skou logic)
Let us show how Larsen-Skou logic works by considering the reactive system in Figure 14.1. We would like to prove the following formula:

$$s_1 \models \langle 1 \in \rangle_{1/2} \langle \textbf{coffee} \rangle \, \textbf{true}$$

By using the semantics we should have:

$$\alpha \, s_1 \, 1 \in I_1 \geq 1/2$$

where:

$$I_1 = \{s \in S \mid s \models \langle \textbf{coffee} \rangle \, \textbf{true}\}$$

Therefore:

$$I_1 = \{s \in S \mid \alpha s \, \textbf{coffee} \, I_2 \geq 1\}$$

where:

$$I_2 = \{s \in S \mid s \models \textbf{true}\} = \{s_1, s_2, s_3\}$$

So we have:

$$I_1 = \{s \in S \mid \alpha s \, \textbf{coffee}\{s_1, s_2, s_3\} \geq 1\} = \{s_2\}$$

Finally :

$$\alpha \, s_1 \, 1 \in \{s_2\} \geq 1/2$$

14.1.2. DTMC With Non-determinism

In this section we will add non-determinism to generative and reactive systems. In this case we use non-determinism to allow the system to choose between different probability distributions. We will introduce two classes of models called *Segala automata* and *simple Segala automata*.

14.1.2.1. Segala Automata

Segala automata have been developed by Roberto Segala in 1995. They are generative systems that combine probability and non-determinism. When the system has to pass from a state to another, first of all it has to choose non-deterministically a probability distribution, then it uses this information to perform the transition. Formally the transition function is defined as follows:

$$\alpha_s : S \rightarrow \mathscr{P}(D(L \times S))$$

As we can see, the distribution is defined on pairs of labels and states. Note that in this case it is not necessary to have the singleton to represent a deadlock state since we can use the empty set.

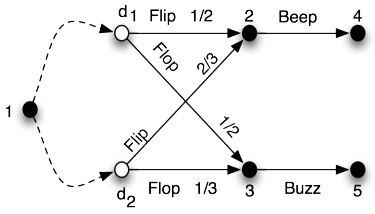


Figure 14.2.: A Segala automata

Example 14.4 (Segala automata)
Let us consider the system in Figure 14.2. We have an automata with five states (black dots). When it is in the state 1 *the system can choose non-deterministically (dotted arrows) between two different distributions* d_1 *and* d_2*:*

$$\alpha_S(1) = \{d_1, d_2\} \quad \textit{where} \quad \begin{array}{ll} d_1(Flip, 2) = \frac{1}{2} & d_1(Flop, 3) = \frac{1}{2} \\ d_2(Flip, 2) = \frac{2}{3} & d_2(Flop, 3) = \frac{1}{3} \end{array}$$

$$\alpha_S(2) = \{d_3\} \quad \textit{where } d_3(Beep, 4) = 1$$

$$\alpha_S(3) = \{d_4\} \quad \textit{where } d_4(Buzz, 5) = 1$$

$$\alpha_S(4) = \alpha_S(5) = \varnothing$$

14.1.2.2. Simple Segala Automata

Now we present the reactive version of Segala automata. In this case we have that the system can react to an external stimulation by using a probability distribution. Since we can have more than one distribution for each label, the system uses non-determinism to choose between different distributions for the same label. Formally a *simple Segala automata* is defined as follows:

$$\alpha_{simS} : S \to \mathscr{P}(L \times D(S))$$

Example 14.5 (A Simple Segala Automata)
Let us consider the system in Figure14.3 (for some suitable probability value ϵ*). We have six states (black dots), the state* 1 *has two possible inputs, a and c, moreover the label a has two different distributions* d_1 *and* d_3*. Formally the system is defined as follows:*

$$\alpha_{simS}(1) = \{(a, d_1), (c, d_2), (a, d_3)\} \textit{ where}$$

$$\begin{array}{ll} d_1(2) = d_1(3) = \frac{1}{2} & \\ d_2(4) = \frac{1}{3} & d_2(5) = \frac{2}{3} \\ d_3(1) = \epsilon & d_3(6) = 1 - \epsilon \end{array}$$

$$\alpha_{simS}(2) = \alpha_{simS}(3) = \alpha_{simS}(4) = \alpha_{simS}(5) = \alpha_{simS}(6) = \varnothing$$

14.1.2.3. Non-determinism, Probability and Actions

As we saw there are many ways to combine probability, non-determinism and actions. We conclude this chapter by mentioning two other interesting models which can be obtained by redefining the transition

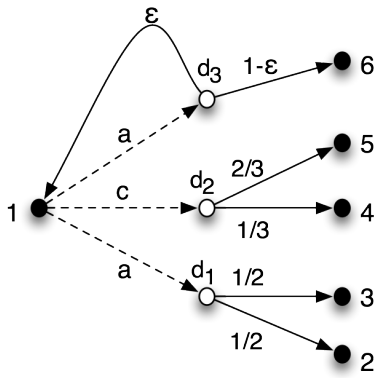


Figure 14.3.: A Simple Segala automata

function of a PTS.

The first class of systems is that of *alternating* transition systems. In this case we allow the system to perform two types of transition: one using probability distributions and one using non-determinism. An alternating system can be defined formally as follows:

$$\alpha : S \longrightarrow (D(S) + \mathscr{P}(L \times S))$$

So in this kind of systems we can alternate probabilistic and non-deterministic choices.

The second type of systems that we present is that of *bundle* transition systems. In this case the system associates a distribution to subsets of non-deterministic choices. Formally:

$$\alpha : S \longrightarrow D(\mathscr{P}(L \times S))$$

So when a bundle transition system has to perform a transition, first of all it chooses by using a probability distribution a set of possible choices, then non-deterministically it picks one of these.

15. PEPA - Performance Evaluation Process Algebra

In this last chapter we introduce a language for quantitative analysis of systems called PEPA. To understand the differences between qualitative analysis and quantitative analysis, we remark that qualitative questions like:

- Will the system arrive in a particular state?
- Does the system behaviour match its specification?
- Does a given property ϕ hold within the system?

are replaced by quantitative questions like:

- How long will it take for the system to arrive in a particular state?
- With what probability does the system behaviour match its specification?
- Does a given property ϕ hold within the system within time t with probability p?

Jane Hillston defined the PEPA language in her PhD thesis in 1994. This process algebra has been developed as a high-level language for the description of continuous time Markov chains. It was obtained by extending CSP (Calculus for Sequential Processes) with probabilities. Over the years PEPA has been shown to provide an expressive formal language for modelling distributed systems. In the spirit of process algebras, PEPA models are obtained as the structure assembly of components that perform individual activities at certain rates and can cooperate on shared actions. The most important features of PEPA w.r.t. other approaches to performance modelling are:

- compositionality, i.e., the ability to model a system as the interaction of subsystems;
- formality, i.e., rigorous semantics giving a precise meaning to all terms in the language; and
- abstraction, i.e., the ability to build up complex models from detailed components, disregarding the details when it is appropriate to do so;
- separation of concerns, i.e., the ability to model the components and the interaction separately;
- structure, i.e., the ability to impose a clear structure to models, which become easy to understand;
- refinement, i.e., the ability to construct models systematically by refining their specifications;
- reusability, i.e., the ability to maintain a library of model components.

For example, queueing networks offer compositionality but not formality; stochastic extensions of Petri nets offer formality but not compositionality; neither offer abstraction mechanisms.

We will start with a brief introduction to CSP, then we will conclude with the presentation of PEPA.

15.1. CSP

Communicating Sequential Processes (CSP) is a process algebra introduced by C. A. R. Hoare in 1978 and is a very powerful tool for systems specification and verification. Contrary to CCS, CSP actions have no dual counterpart and the synchronization between two or more processes is possible when they all perform the same action α (in which case the result of the synchronization is still α). Since during communication the joint action remains visible to the environment, it can be used to interact with other (more than two) processes, realizing multiway synchronization.

15.1.1. Syntax of CSP

The syntax of CSP is the following:

$$P, Q ::= \mathbf{nil} \mid \alpha.P \mid P + Q \mid P \underset{L}{\bowtie} Q \mid P/L \qquad \text{Where } L \text{ is a set of actions}$$

nil : is the inactive process.

$\alpha.P$: is a process which can perform an action α and then behaves like P.

$P + Q$: is a process which can choose to behave like P or like Q.

$P \underset{L}{\bowtie} Q$: is a synchronization operator, also called *cooperation combinator*. It is more precisely an indexed family of operators, one for each possible set of actions L. The set L is called *cooperation set* and fixes the set of *shared actions* between P and Q. Processes P and Q can use the actions in L to synchronize each other. The actions not included in L are called *individual activities* and can be performed separately by P and Q. As a special case, if $L = \varnothing$ then all the actions of P and Q are just interleaved. Note that this operator is not associative, for example $(\alpha.\beta.\mathbf{nil} \underset{\{\alpha\}}{\bowtie} \mathbf{nil}) \underset{\varnothing}{\bowtie} \alpha.\mathbf{nil} \neq \alpha.\beta.\mathbf{nil} \underset{\{\alpha\}}{\bowtie} (\mathbf{nil} \underset{\varnothing}{\bowtie} \alpha.\mathbf{nil})$, in fact the first process can perform only an action α, the second process on the contrary can perform a synchronization on α reaching the state $\beta.\mathbf{nil} \underset{\{\alpha\}}{\bowtie} (\mathbf{nil} \underset{\varnothing}{\bowtie} \mathbf{nil})$ then it can perform an action β reaching $\mathbf{nil} \underset{\{\alpha\}}{\bowtie} (\mathbf{nil} \underset{\varnothing}{\bowtie} \mathbf{nil})$.

P/L : is the hiding operator. If an action α is in L then it can be performed only as a silent action τ.

15.1.2. Operational Semantics of CSP

Now we present the semantics of CSP. As we have done for CCS we define the semantics of CSP by using a rule system.

$$\frac{}{\alpha.P \xrightarrow{\alpha} P} \qquad \frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'} \qquad \frac{Q \xrightarrow{\alpha} Q'}{P + Q \xrightarrow{\alpha} Q'}$$

$$\frac{P \xrightarrow{\alpha} P' \quad \alpha \notin L}{P/L \xrightarrow{\alpha} P'/L} \qquad \frac{P \xrightarrow{\alpha} P' \quad \alpha \in L}{P/L \xrightarrow{\tau} P'/L}$$

$$\frac{P \xrightarrow{\alpha} P' \quad \alpha \notin L}{P \underset{L}{\bowtie} Q \xrightarrow{\alpha} P' \underset{L}{\bowtie} Q} \qquad \frac{Q \xrightarrow{\alpha} Q' \quad \alpha \notin L}{P \underset{L}{\bowtie} Q \xrightarrow{\alpha} P \underset{L}{\bowtie} Q'} \qquad \frac{P \xrightarrow{\alpha} P' \quad Q \xrightarrow{\alpha} Q' \quad \alpha \in L}{P \underset{L}{\bowtie} Q \xrightarrow{\alpha} P' \underset{L}{\bowtie} Q'}$$

15.2. PEPA

As we said PEPA is obtained by adding probabilities to CSP. As we will see there are not explicit probabilistic operators in PEPA, but the probabilistic behaviour is obtained by associating an exponentially distributed continuous random variable to each action, this random variable will represent the time needed to execute the action. These random variables lead to a clear relationship between the process algebra model and a CTMC. Via this underlying Markov process performance measures can then be extracted from the model.

15.2.1. Syntax of PEPA

In PEPA an action is a pair (α, r) where α is the action type and r is the rate of the continuous random variable associated with the action. The rate r can be any positive real number. We have the following grammar:

$$P, Q ::= \mathbf{nil} \mid (\alpha, r).P \mid P + Q \mid P \underset{L}{\bowtie} Q \mid P/L \mid C$$

$(\alpha, r).P$: is a process which can perform an action α and then behaves like P. In this case the rate r is used to define the exponential variable which describes the duration of the action. A component may have a purely sequential behaviour, repeatedly undertaking one activity after another and eventually returning to its initial state. As a simple example, consider a web server in a distributed system that can serve one request at a time:

$$WS \stackrel{\text{def}}{=} (request, \top).(serve, \mu).(respond, \top).WS$$

In some cases, as here, the rate of an action falls out of the control of this component: such actions are carried out jointly with another component, with the current component playing some sort of passive role. For example, the web server is passive with respect to the request and respond actions, as it cannot influence the rate at which application execute these actions. This is recorded by using the distinguished rate $\top$ which we can assume to represent an extremely high value that cannot influence the rate of interacting components.

$P + Q$: has the same meaning of the CSP operator for choice. For example, we can consider an application in a distributed system that can either access a locally available method (with probability p_1) or access to a remote web service (with probability $p_2 = 1 - p_1$). The decision is taken by performing a think action which is parametric to the rate λ:

$$\begin{aligned} Appl \stackrel{\text{def}}{=} & (think, p_1 \cdot \lambda).(local, m).Appl \\ + & (think, p_2 \cdot \lambda).(request, rq).(respond, rp).Appl \end{aligned}$$

$P \underset{L}{\bowtie} Q$: has the same meaning of the CSP operator. In the web service example, we can assume that the application and the web server interact overthe set of shared actions $L = \{request, respond\}$:

$$Sys \stackrel{\text{def}}{=} (Appl \underset{\varnothing}{\bowtie} Appl) \underset{L}{\bowtie} WS$$

During the interaction, the resulting action will have the same type of the shared action and a rate reflecting the rate of the slowest action.

P/L : is the same of the CSP hiding operator: the duration of the action is unaffected, but its type becomes hidden. In our running example, we may want to hide the local computation of $Appl$ to the environment:

$$Appl' \stackrel{\text{def}}{=} Appl/\{local\}$$

C : is the name of a recursive process which is defined as $C \stackrel{\text{def}}{=} P$ in a separate set Δ of declarations. Using recursive definition as the ones given above for $Appl$ and WS we are able to describe components with infinite behaviour without introducing an explicit recursion or replication operator.

Usually we are interested only in those agents which have an ergodic underlying Markov process, this because we want to apply the steady state analysis. It has been shown that it is possible to ensure the ergodicity by using syntactic restrictions on the agents. In particular, the class of PEPA terms which satisfy these syntactic conditions are called *cyclic components* and they can be described by the following grammar:

$$\begin{aligned} P, Q & ::= S \mid P \underset{L}{\bowtie} Q \mid P/L \\ S, T & ::= (\alpha, r).S \mid S + T \mid C \end{aligned}$$

where each recursive process C is sequential, i.e., $C \stackrel{\text{def}}{=} S$ for some sequential process S.

15.2.2. Operational Semantics of PEPA

We give the PEPA operational semantics by using the following rule system whose well formed formulas have the form $P \xrightarrow{(\alpha,r)} Q$ for suitable PEPA processes P and Q, activity α and rate r. We assume a set Δ of declarations.

$$\frac{}{(\alpha, r).P \xrightarrow{(\alpha,r)} P} \qquad \frac{P \xrightarrow{(\alpha,r)} P'}{P + Q \xrightarrow{(\alpha,r)} P'} \qquad \frac{Q \xrightarrow{(\alpha,r)} Q'}{P + Q \xrightarrow{(\alpha,r)} Q'}$$

$$\frac{P \xrightarrow{(\alpha,r)} P' \quad \alpha \notin L}{P/L \xrightarrow{(\alpha,r)} P'/L} \qquad \frac{P \xrightarrow{(\alpha,r)} P' \quad \alpha \in L}{P/L \xrightarrow{(\tau,r)} P'/L}$$

$$\frac{P \xrightarrow{(\alpha,r)} P' \quad \alpha \notin L}{P \underset{L}{\bowtie} Q \xrightarrow{(\alpha,r)} P' \underset{L}{\bowtie} Q} \qquad \frac{Q \xrightarrow{(\alpha,r)} Q' \quad \alpha \notin L}{P \underset{L}{\bowtie} Q \xrightarrow{(\alpha,r)} P \underset{L}{\bowtie} Q'}$$

$$\frac{P \xrightarrow{(\alpha,r_1)} P' \quad Q \xrightarrow{(\alpha,r_2)} Q' \quad \alpha \in L}{P \underset{L}{\bowtie} Q \xrightarrow{(\alpha,r)} P' \underset{L}{\bowtie} Q'} \quad \text{where } r = \min(r_\alpha(P), r_\alpha(Q)) * \frac{r_1}{r_\alpha(P)} * \frac{r_2}{r_\alpha(Q)}$$

$$\frac{P \xrightarrow{(\alpha,r)} P' \quad (C \stackrel{\text{def}}{=} P) \in \Delta}{C \xrightarrow{(\alpha,r)} P'}$$

The only rule that deserves some explanation is the cooperation rule, where we have denoted by $r_\alpha(P)$ the *apparent rate* of action α in P, which is defined by structural recursion as follows:

$$r_\alpha((\beta, r).P) = \begin{cases} r & \text{if } \alpha = \beta \\ 0 & \text{if } \alpha \neq \beta \end{cases}$$

$$r_\alpha(P + Q) = r_\alpha(P) + r_\alpha(Q)$$

$$r_\alpha(P/L) = \begin{cases} r_\alpha(P) & \text{if } \alpha \notin L \\ 0 & \text{if } \alpha \in L \end{cases}$$

$$r_\alpha(P \underset{L}{\bowtie} Q) = \begin{cases} min(r_\alpha(P), r_\alpha(Q)) & \text{if } \alpha \in L \\ r_\alpha(P) + r_\alpha(Q) & \text{if } \alpha \notin L \end{cases}$$

Roughly, the apparent rate $r_\alpha(S)$ is the sum of the rates of all distinct actions α that can be performed by S, thus $r_\alpha(S)$ expresses the overall rate of α in S (because of the property of rates of exponentially distributed variables). Notably, in the case of shared actions $P \underset{L}{\bowtie} Q$ the apparent rate is the slower of the apparent rates of P and Q.

The selection of the exponential distribution as the governing distribution for action durations in PEPA has profound consequences. In terms of the underlying stochastic process, it is the only choice which gives rise to a Markov process. This is due to the memoryless properties of the exponential distribution: the time until the next event is independent of the time since the last event, because the exponential distribution forgets how long it has already waited. For instance, if we consider the process $(\alpha, r).\,\mathbf{nil} \underset{\varnothing}{\bowtie} (\beta, s).\,\mathbf{nil}$ and the system performs the action α, the time needed to complete β from $\mathbf{nil} \underset{\varnothing}{\bowtie} (\beta, s).\,\mathbf{nil}$ does not need to consider the time already taken to carry out the action α.

Let us now explain the calculation $r = \min(r_\alpha(P), r_\alpha(Q)) * \frac{r_1}{r_\alpha(P)} * \frac{r_2}{r_\alpha(Q)}$ that appears in the cooperation rule. The best way to resolve what should be the rate of the shared action has been a topic of some debate. The definition of cooperation in PEPA is based on the assumption that a component cannot be made to exceed its bounded capacity for carrying out the shared actions, where the bounded capacity consists of the apparent rate of the action. The underlying assumption is that the choice of a specific action (with rate r_i) to carry on the shared activity occurs independently in the two cooperating components P and Q. Now, the probability that a certain action (α, r_1) (respectively, (α, r_2)) is chosen by P is $\frac{r_1}{r_\alpha(P)}$ (respectively, $\frac{r_2}{r_\alpha(Q)}$ in the case of Q). Then, from the independence of the choices in P and Q we obtain the combined probability $\frac{r_1}{r_\alpha(P)} * \frac{r_2}{r_\alpha(Q)}$. Finally, the resulting rate is the product of the apparent rate $\min(r_\alpha(P), r_\alpha(Q))$ and the above probability. Notice that if we sum up the rates of all possible synchronizations on α of $P \underset{L}{\bowtie} Q$ we just get $min(r_\alpha(P), r_\alpha(Q))$ as stated above. See the example below.

Example 15.1
Let us define two PEPA agents as follows:

$$P = (\alpha, r).P_1 + \ldots + (\alpha, r).P_n$$

$$Q = (\alpha, r).Q_1 + \ldots + (\alpha, r).Q_m$$

We assume $n \leq m$:

$$P \xrightarrow{(\alpha,r)} P_i \qquad i = 1, \ldots, n$$

$$Q \xrightarrow{(\alpha,r)} Q_i \qquad i = 1, \ldots, m$$

So we have the the following apparent rates:

$$r_\alpha(P) = n * r$$

$$r_\alpha(Q) = m * r$$

Then considering the synchronization between P and Q, we have:

$$P \underset{\{\alpha\}}{\bowtie} Q \xrightarrow{(\alpha,r')} P_i \underset{\{\alpha\}}{\bowtie} Q_j \qquad i = 0 \ldots n \qquad j = 1 \ldots m$$

Where $r' = \frac{r}{nr} * \frac{r}{mr} * nr = \frac{r}{m}$. The apparent rate of the synchronization is:

$$r_\alpha(P \underset{\{\alpha\}}{\bowtie} Q) = m * n * r' = m * n * \frac{r}{m} = n * r = min(r_\alpha(P), r_\alpha(Q))$$

The underlying CTMC is obtained from the LTS by associating a state with each process, and the transitions between states are derived from the transitions of the LTS. Since all activity durations are exponentially distributed, the total transition rate between two states will be the sum of the activity rates labelling arcs connecting the corresponding nodes in the LTS.

We conclude this section by showing an example of process modelled by using PEPA.

Example 15.2 (Roland the gunman)
We want to model a far west duel. We have two main characters: Roland the gunman and his enemies. Upon its travels Roland will encounter some enemies with whom he will have no choice but to fight back. For simplicity we assume that Roland has two guns with one bullet in each and that each hit is fatal. We also assume that a sense of honour prevents an enemy from attacking Roland if he is already involved in a gun fight. We model the behaviour of Roland as follows. Normally, Roland is in an idle state $Roland_{idle}$, but when he is attacked (attacks) he moves to state $Roland_2$, where he has two bullets available in his gun:

$$Roland_{idle} \stackrel{\text{def}}{=} (attack, \top).Roland_2$$

In front of his enemies, Roland can act in three ways: if he hits his enemies then he reloads his gun and returns idle; if he misses the enemies he tries a second attack (see $Roland_1$); finally if an enemy hits him, Roland dies.

$$\begin{aligned} Roland_2 \stackrel{\text{def}}{=} & (hit, r_{hit}).(reload, r_{reload}).Roland_{idle} \\ & +(miss, r_{miss}).Roland_1 \\ & +(e\text{-}hit, \top).Roland_{dead} \end{aligned}$$

The second attempt to shoot by Roland is analogous to the first one, but this time it is the last bullet in Roland's gun and if the enemy is missed no further shot is possible in $Roland_{empty}$ until the gun is reloaded.

$$\begin{aligned} Roland_1 \stackrel{\text{def}}{=} & (hit, r_{hit}).(reload, r_{reload}).Roland_{idle} \\ & +(miss, r_{miss}).Roland_{empty} \\ & +(e\text{-}hit, \top).Roland_{dead} \\ Roland_{empty} \stackrel{\text{def}}{=} & (reload, r_{reload}).Roland_2 \\ & +(e\text{-}hit, \top).Roland_{dead} \end{aligned}$$

Finally if Roland is dead he cannot perform any action.

$$Roland_{dead} \stackrel{\text{def}}{=} \mathbf{nil}$$

We describe enemies behaviour as follows. If the enemies are idle they can try to attack Roland:

$$Enemies_{idle} \stackrel{\text{def}}{=} (attack, r_{attack}).Enemies_{attack}$$

Enemies shoot once and either get hit or they hit Roland.

$$\begin{aligned} Enemies_{attack} \stackrel{\text{def}}{=} & (e\text{-}hit, r_{e\text{-}hit}).Enemies_{idle} \\ & +(hit, \top).Enemies_{idle} \end{aligned}$$

The rates involved in the model are measured in seconds, so a rate of 1.0 *would indicate that the action is expected to occur once every second. We define the following rates:*

$\top$	$=$ *about* ∞	
r_{fire}	$= 1$	*one shot per second*
$r_{hit\text{-}success}$	$= 0.8$	*80% of success*
r_{hit}	$= 0.8$	$r_{fire} * r_{hit\text{-}success}$
r_{miss}	$= 0.2$	$r_{fire} * (1 - r_{hit\text{-}success})$
r_{reload}	$= 0.3$	*3 seconds to reload*
r_{attack}	$= 0.01$	*Roland is attacked once every 100 seconds*
$r_{e\text{-}hit}$	$= 0.02$	*Enemies can hit once every 50 seconds*

So we model the duel as follows:

$$Duel \stackrel{\text{def}}{=} Roland_{idle} \underset{\{hit, attack, e-hit\}}{\bowtie} Enemies_{idle}$$

We can perform various types of analysis of the system by using standard methods. Using the steady state analysis, that we have seen in the previous chapters, we can prove that Roland will always die and the system will deadlock, because there is an infinite supply of enemies (so the system is not ergodic). Moreover we can answer many other questions by using the following techniques:

- *Transient analysis: we can ask for the probability that Roland is dead after 1 hour, or the probability that Roland will have killed some enemy within 30 minutes.*
- *Passage time analysis: we can ask for the probability of passing at least 10 seconds from the first attack to Roland to the time it has hit 3 enemies, or the probability that 1 minute after he is attacked Roland has killed his attacker (i.e., the probability that the model performs a hit action within 1 minute after having performed an attack action).*

The PEPA language is supported by a range of tools and by a wide community of users. PEPA application areas span the subject areas of informatics and engineering including, e.g., cellular telephone networks, database systems, diagnostic expert systems, multiprocessor access-contention protocols, protocols for fault-tolerant systems, software architectures. Additional information and a PEPA Eclipse Plug-in are freely available at `http://www.dcs.ed.ac.uk/pepa/`.

Part VII.

Appendices

A. Summary

A.1. Induction rules 3.1.2

A.1.1. Noether

Let $\prec$ be a well-founded relation over the set A and let P be a unary predicate over A. Then:

$$\frac{\forall a \in A.\ (\forall b \prec a.\ P(b)) \rightarrow P(a)}{\forall a \in A.P(a)}$$

A.1.2. Weak Mathematical Induction 3.1.3

Let P be a unary predicate over ω.

$$\frac{P(0) \quad \forall n \in \omega.(P(n) \rightarrow P(n+1))}{\forall n \in \omega.P(n)}$$

A.1.3. Strong Mathematical Induction 3.1.4

Let P be a unary predicate over ω.

$$\frac{P(0) \quad \forall n \in \omega.(\forall i \leq n.P(i)) \rightarrow P(n+1)}{\forall n \in \omega.P(n)}$$

A.1.4. Structural Induction 3.1.5

Let Σ be a signature, T_Σ be the set of terms over Σ and P be a property defined on T_Σ.

$$\frac{\forall t \in T_\Sigma.\ (\forall t' < t.\ P(t')) \Rightarrow P(t)}{\forall t \in T_\Sigma.\ P(t)}$$

A.1.5. Derivation Induction 3.1.6

Let R be a set of inference rules and D the set of derivations defined on R. We define:

$$\frac{\forall \{x_1, \ldots, x_n\}/y \in R.\ (P(d_1) \wedge \ldots \wedge P(d_n)) \Rightarrow P(\{d_1, \ldots, d_n\}/y)}{\forall d \in D.\ P(d)}$$

where $d_1, \ldots, d_n$ are derivation for $x_1, \ldots, x_n$.

A.1.6. Rule Induction 3.1.7

Let R be a set of rules and I_R the set of theorems of R.

$$\frac{\forall (X/y) \in R \quad X \subseteq I_R \quad (\forall x \in X \cdot P(x)) \Longrightarrow P(y)}{\forall x \in I_R.P(x)}$$

A.1.7. Computational Induction 5.4

Let P be a property, $(D, \sqsubseteq)$ a $CPO_\perp$ and F a monotone, continuous function on it. We define:

$$\frac{P \text{ inclusive} \quad \perp \in P \quad \forall d \in D.d \in P \Longrightarrow F(d) \in P}{fix(F) \in P}$$

A.2. IMP 2

A.2.1. IMP Syntax 2.1

$$\begin{array}{rcl} a & ::= & n \mid x \mid a_0 + a_1 \mid a_0 - a_1 \mid a_0 \times a_1 \\ b & ::= & v \mid a_0 = a_1 \mid a_0 \leq a_1 \mid \neg b \mid b_0 \vee b_1 \mid b_0 \wedge b_1 \\ c & ::= & \textbf{skip} \mid x := a \mid c_0; c_1 \mid \textbf{if } b \textbf{ then } c_0 \textbf{ else } c_1 \mid \textbf{while } b \textbf{ do } c \end{array}$$

A.2.2. IMP Operational Semantics 2.2

A.2.2.1. IMP Arithmetic Expressions

$$\frac{}{\langle x, \sigma\rangle \to \sigma(x)}\ (\text{ide}) \qquad \frac{}{\langle n, \sigma\rangle \to n}\ (\text{num}) \qquad \frac{\langle a_0, \sigma\rangle \to n_0 \quad \langle a_1, \sigma\rangle \to n_1}{\langle a_0 + a_1, \sigma\rangle \to n_0 + n_1}\ (\text{sum})$$

$$\frac{\langle a_0, \sigma\rangle \to n_0 \quad \langle a_1, \sigma\rangle \to n_1}{\langle a_0 - a_1, \sigma\rangle \to n_0 - n_1}\ (\text{dif}) \qquad \frac{\langle a_0, \sigma\rangle \to n_0 \quad \langle a_1, \sigma\rangle \to n_1}{\langle a_0 \times a_1, \sigma\rangle \to n_0 \times n_1}\ (\text{prod})$$

A.2.2.2. IMP Boolean Expressions

$$\frac{}{\langle v, \sigma\rangle \to v}\ (\text{bool}) \qquad \frac{\langle a_0, \sigma\rangle \to n_0 \quad \langle a_1, \sigma\rangle \to n_1}{\langle a_0 = a_1, \sigma\rangle \to (n_0 = n_1)}\ (\text{equ}) \qquad \frac{\langle a_0, \sigma\rangle \to n_0 \quad \langle a_1, \sigma\rangle \to n_1}{\langle a_0 \leq a_1, \sigma\rangle \to (n_0 \leq n_1)}\ (\text{leq})$$

$$\frac{\langle b, \sigma\rangle \to v}{\langle \neg b, \sigma\rangle \to \neg v}\ (\text{not}) \qquad \frac{\langle b_0, \sigma\rangle \to v_0 \quad \langle b_1, \sigma\rangle \to v_1}{\langle b_0 \vee b_1, \sigma\rangle \to (v_0 \vee v_1)}\ (\text{or}) \qquad \frac{\langle b_0, \sigma\rangle \to v_0 \quad \langle b_1, \sigma\rangle \to v_1}{\langle b_0 \wedge b_1, \sigma\rangle \to (v_0 \wedge v_1)}\ (\text{and})$$

A.2.2.3. IMP Commands

$$\frac{}{\langle \textbf{skip}, \sigma\rangle \to \sigma}\ (\text{skip}) \qquad \frac{\langle a, \sigma\rangle \to m}{\langle x := a, \sigma\rangle \to \sigma[{}^{m}/_{x}]}\ (\text{assign})$$

$$\frac{\langle c_0, \sigma\rangle \to \sigma'' \quad \langle c_1, \sigma''\rangle \to \sigma'}{\langle c_0; c_1, \sigma\rangle \to \sigma'}\ (\text{seq}) \qquad \frac{\langle b, \sigma\rangle \to \textbf{true} \quad \langle c_0, \sigma\rangle \to \sigma'}{\langle \textbf{if } b \textbf{ then } c_0 \textbf{ else } c_1, \sigma\rangle \to \sigma'}\ (\text{iftt}) \qquad \frac{\langle b, \sigma\rangle \to \textbf{false} \quad \langle c_1, \sigma\rangle \to \sigma'}{\langle \textbf{if } b \textbf{ then } c_0 \textbf{ else } c_1, \sigma\rangle \to \sigma'}\ (\text{ifff})$$

$$\frac{\langle b, \sigma\rangle \to \textbf{true} \quad \langle c, \sigma\rangle \to \sigma'' \quad \langle \textbf{while } b \textbf{ do } c, \sigma''\rangle \to \sigma'}{\langle \textbf{while } b \textbf{ do } c, \sigma\rangle \to \sigma'}\ (\text{whtt}) \qquad \frac{\langle b, \sigma\rangle \to \textbf{false}}{\langle \textbf{while } b \textbf{ do } c, \sigma\rangle \to \sigma}\ (\text{whff})$$

A.2.3. IMP Denotational Semantics 5

A.2.3.1. IMP Arithmetic Expressions $\mathscr{A} : Aexpr \to (\Sigma \to \mathbb{N})$

$$\begin{array}{rcl} \mathscr{A}\,[\![n]\!]\,\sigma & = & n \\ \mathscr{A}\,[\![x]\!]\,\sigma & = & \sigma x \\ \mathscr{A}\,[\![a_0 + a_1]\!]\,\sigma & = & (\mathscr{A}\,[\![a_0]\!]\,\sigma) + (\mathscr{A}\,[\![a_1]\!]\,\sigma) \\ \mathscr{A}\,[\![a_0 - a_1]\!]\,\sigma & = & (\mathscr{A}\,[\![a_0]\!]\,\sigma) - (\mathscr{A}\,[\![a_1]\!]\,\sigma) \\ \mathscr{A}\,[\![a_0 \times a_1]\!]\,\sigma & = & (\mathscr{A}\,[\![a_0]\!]\,\sigma) \times (\mathscr{A}\,[\![a_1]\!]\,\sigma) \end{array}$$

A.2.3.2. IMP Boolean Expressions $\mathscr{B} : Bexpr \to (\Sigma \to \mathbb{B})$

$$\begin{aligned}
\mathscr{B}[\![v]\!]\sigma &= v \\
\mathscr{A}[\![a_0 = a_1]\!]\sigma &= (\mathscr{A}[\![a_0]\!]\sigma) = (\mathscr{A}[\![a_1]\!]\sigma) \\
\mathscr{A}[\![a_0 \leq a_1]\!]\sigma &= (\mathscr{A}[\![a_0]\!]\sigma) \leq (\mathscr{A}[\![a_1]\!]\sigma) \\
\mathscr{B}[\![\neg b_0]\!]\sigma &= \neg(\mathscr{B}[\![b]\!]\sigma) \\
\mathscr{B}[\![b_0 \vee b_1]\!]\sigma &= (\mathscr{B}[\![b_0]\!]\sigma) \vee (\mathscr{B}[\![b_1]\!]\sigma) \\
\mathscr{B}[\![b_0 \wedge b_1]\!]\sigma &= (\mathscr{B}[\![b_0]\!]\sigma) \wedge (\mathscr{B}[\![b_1]\!]\sigma)
\end{aligned}$$

A.2.3.3. IMP Commands $\mathscr{C} : Com \to (\Sigma \rightharpoonup \Sigma)$

$$\begin{aligned}
\mathscr{C}[\![\mathbf{skip}]\!]\sigma &= \sigma \\
\mathscr{C}[\![x := a]\!]\sigma &= \sigma\left[{}^{\mathscr{A}[\![a]\!]\sigma}/_{x}\right] \\
\mathscr{C}[\![c_1; c_2]\!]\sigma &= \mathscr{C}[\![c_2]\!]^*(\mathscr{C}[\![c_1]\!]\sigma) \\
\mathscr{C}[\![\mathbf{if}\, b\, \mathbf{then}\, c_1\, \mathbf{else}\, c_2]\!]\sigma &= \mathscr{B}[\![b]\!]\sigma \to \mathscr{C}[\![c_1]\!]\sigma, \mathscr{C}[\![c_2]\!]\sigma \\
\mathscr{C}[\![\mathbf{while}\, b\, \mathbf{do}\, c]\!] &= fix\,\Gamma = \bigsqcup_{n\in\omega} \Gamma^n(\perp_{\Sigma\to\Sigma_\perp})
\end{aligned}$$

where Γ is defined as follows:

$$\Gamma\varphi\sigma \quad = \quad \mathscr{B}[\![\mathbf{true}]\!]\sigma \to \varphi^*(\mathscr{C}[\![\mathbf{skip}]\!]\sigma), \sigma$$

A.3. HOFL 6.1

A.3.1. HOFL Syntax 6.1

$t ::=$	$x \mid$		Variable
	$n \mid$		Constant
	$t_1 + t_2 \mid t_1 - t_2 \mid t_1 \times t_2 \mid$		Arithmetic Operators
	$\mathbf{if}\, t\, \mathbf{then}\, t_1\, \mathbf{else}\, t_2 \mid$		Conditional
	$(t_1, t_2) \mid \mathbf{fst}(t) \mid \mathbf{snd}(t) \mid$		Pairing and Projection Operators
	$\lambda x.t \mid (t_1\ t_2) \mid$		Function Abstraction and Application
	$\mathbf{rec}\ x.t$		Recursion

A.3.2. HOFL Types 6.1.1

$$\tau ::= int \mid \tau * \tau \mid \tau \to \tau$$

A.3.3. HOFL Typing Rules 6.1.1

$$n : int \qquad \frac{t_1 : int \quad t_2 : int}{t_1\ \mathrm{op}\ t_2 : int}\ \text{with op} = +, -, \text{x} \qquad \frac{t_0 : int \quad t_1 : \tau \quad t_2 : \tau}{\mathbf{if}\, t_0\, \mathbf{then}\, t_1\, \mathbf{else}\, t_2 : \tau}$$

$$\frac{t_1 : \tau_1 \quad t_2 : \tau_2}{(t_1, t_2) : \tau_1 * \tau_2} \qquad \frac{t : \tau_1 * \tau_2}{\mathbf{fst}(t) : \tau_1} \qquad \frac{t : \tau_1 * \tau_2}{\mathbf{snd}(t) : \tau_2}$$

$$\frac{x : \tau_1 \quad t : \tau_2}{\lambda x.t : \tau_1 \to \tau_2} \qquad \frac{t_1 : \tau_1 \to \tau_2 \qquad t_2 : \tau_1}{(t_1\ t_2) : \tau_2}$$

$$\frac{x : \tau \quad t : \tau}{\mathbf{rec}\ x.t : \tau}$$

A.3.4. HOFL Operational Semantics 6.2

A.3.4.1. HOFL Canonical Forms

$$\frac{\varnothing}{n \in C_{int}} \qquad \frac{t_1 : \tau_1 \quad t_2 : \tau_2 \quad t_1, t_2 \text{ closed}}{(t_1, t_2) \in C_{\tau_1 * \tau_2}} \qquad \frac{\lambda x.t : \tau_1 \to \tau_2 \quad \lambda x.t \text{ closed}}{\lambda x.t \in C_{\tau_1 \to \tau_2}}$$

A.3.4.2. HOFL Axiom

$$\frac{}{c \to c}$$

A.3.4.3. HOFL Arithmetic and Conditional Expressions

$$\frac{t_1 \to n_1 \quad t_2 \to n_2}{t_1 \text{ op } t_2 \to n_1 \underline{op}\, n_2} \qquad \frac{t_0 \to 0 \quad t_1 \to c_1}{\textbf{if } t_0 \textbf{ then } t_1 \textbf{ else } t_2 \to c_1} \qquad \frac{t_0 \to n \quad n \neq 0 \quad t_2 \to c_2}{\textbf{if } t_0 \textbf{ then } t_1 \textbf{ else } t_2 \to c_2}$$

A.3.4.4. HOFL Pairing Rules

$$\frac{t \to (t_1, t_2) \quad t_1 \to c_1}{\textbf{fst}(t) \to c_1} \qquad \frac{t \to (t_1, t_2) \quad t_2 \to c_2}{\textbf{snd}(t) \to c_2}$$

A.3.4.5. HOFL Function Application

$$\frac{t_1 \to \lambda x.t_1' \quad t_1'[{}^{t_2}/_x] \to c}{(t_1 \quad t_2) \to c} \quad \text{(lazy)}$$

$$\frac{t_1 \to \lambda x.t_1' \quad t_2 \to c_2 \quad t_1'[{}^{c_2}/_x] \to c}{(t_1 \quad t_2) \to c} \quad \text{(eager)}$$

A.3.4.6. HOFL Recursion

$$\frac{t[{}^{\textbf{rec } x.t}/_x] \to c}{\textbf{rec } x.t \to c}$$

A.3.5. HOFL Denotational Semantics 8

$$\begin{aligned}
[\![n]\!]\rho &= \lfloor n \rfloor \\
[\![x]\!]\rho &= \rho x \\
[\![t_1 \textbf{ op } t_2]\!]\rho &= [\![t_1]\!]\rho \; \underline{op}_{\perp} \; [\![t_2]\!]\rho \\
[\![\textbf{if } t_0 \textbf{ then } t_1 \textbf{ else } t_2]\!]\rho &= Cond\,([\![t_0]\!]\rho, [\![t_1]\!]\rho, [\![t_2]\!]\rho) \\
[\![(t_1, t_2)]\!]\rho &= \lfloor ([\![t_1]\!]\rho, [\![t_2]\!]\rho) \rfloor \\
[\![\textbf{fst}(t)]\!]\rho &= \textbf{let } v \Leftarrow [\![t]\!]\rho. \quad \pi_1 v \\
[\![\textbf{snd}(t)]\!]\rho &= \textbf{let } v \Leftarrow [\![t]\!]\rho. \quad \pi_2 v \\
[\![\lambda x.t]\!]\rho &= \left\lfloor \lambda d.\, [\![t]\!]\rho[{}^{d}/_x] \right\rfloor \\
[\![(t_1\ t_2)]\!] &= \textbf{let } \varphi \Leftarrow [\![t_1]\!]\rho.\varphi([\![t_2]\!]\rho) \\
[\![\textbf{rec } x.t]\!]\rho &= \textbf{fix } \lambda d.\, [\![t]\!]\rho[{}^{d}/_x]
\end{aligned}$$

A.4. CCS 10

A.4.1. CCS Syntax 10.1

$$p, q ::= x \quad | \quad \mathbf{nil} \quad | \quad \mu.p \quad | \quad p\backslash\alpha \quad | \quad p[\Phi] \quad | \quad p + p \quad | \quad p \mid p \quad | \quad \mathbf{rec}\, x.p$$

A.4.2. CCS Operational Semantics 10.2

$$\text{(Act)}\ \frac{}{\mu.p \xrightarrow{\mu} p} \qquad \text{(Res)}\ \frac{p \xrightarrow{\mu} q}{p\backslash\alpha \xrightarrow{\mu} q\backslash\alpha}\ \mu \neq \alpha, \bar{\alpha} \qquad \text{(Rel)}\ \frac{p \xrightarrow{\mu} q}{p[\Phi] \xrightarrow{\Phi(\mu)} q[\Phi]}$$

$$\text{(Sum)}\ \frac{p \xrightarrow{\mu} p'}{p + q \xrightarrow{\mu} p'} \qquad \frac{q \xrightarrow{\mu} q'}{p + q \xrightarrow{\mu} q'}$$

$$\text{(Com)}\ \frac{p \xrightarrow{\mu} p'}{p|q \xrightarrow{\mu} p'|q} \qquad \frac{q \xrightarrow{\mu} q'}{p|q \xrightarrow{\mu} p|q'} \qquad \frac{p_1 \xrightarrow{\lambda} p_2 \quad q_1 \xrightarrow{\bar{\lambda}} q_2}{p_1|q_1 \xrightarrow{\tau} p_2|q_2}$$

$$\text{(Rec)}\ \frac{p[\mathbf{rec}\, x.p/x] \xrightarrow{\mu} q}{\mathbf{rec}\, x.p \xrightarrow{\mu} q}$$

A.5. CCS Abstract Semantics 10.3

A.5.0.1. CCS Strong Bisimulation 10.3.3

$$\forall s_1\ R\ s_2 \Rightarrow \begin{array}{ll} \text{if } s_1 \xrightarrow{\alpha} s_1' & \text{then there exists a transition } s_2 \xrightarrow{\alpha} s_2' \text{ such that } s_1'\ R\ s_2' \\ \text{if } s_2 \xrightarrow{\alpha} s_2' & \text{then there exists a transition } s_1 \xrightarrow{\alpha} s_1' \text{ such that } s_1'\ R\ s_2' \end{array}$$

A.5.0.2. CCS Axioms for Strong Bisimilarity 10.6

$$\begin{aligned} p + \mathbf{nil} &= p \\ p_1 + p_2 &= p_2 + p_1 \\ p_1 + (p_2 + p_3) &= (p_1 + p_2) + p_3 \\ p + p &= p \end{aligned}$$

A.5.0.3. CCS Weak Bisimulation 10.7

The weak transition relation $\Rightarrow$is defined as follows:

$$\begin{aligned} p \overset{\tau}{\Rightarrow} q \quad &\text{iff} \quad p \xrightarrow{\tau} \ldots \xrightarrow{\tau} q \ \vee\ p = q \\ p \overset{\lambda}{\Rightarrow} q \quad &\text{iff} \quad p \overset{\tau}{\Rightarrow} p' \xrightarrow{\lambda} q' \overset{\tau}{\Rightarrow} q \end{aligned}$$

A weak bisimulation is defined as follows:

$$p\ \Psi(R)\ q \overset{\text{def}}{=} \begin{cases} p \xrightarrow{\mu} p' \text{ then } \exists q'.\ q \overset{\mu}{\Rightarrow} q' & \text{and} \quad p'\ R\ q' \\ q \xrightarrow{\mu} q' \text{ then } \exists p'.\ p \overset{\mu}{\Rightarrow} p' & \text{and} \quad p'\ R\ q' \end{cases}$$

And we define the *weak bisimilarity* as follows:

$$p \approx q \Leftrightarrow \exists R.\ p\ R\ q \wedge \Psi(R) \sqsubseteq R$$

A.5.0.4. CCS Observational Congruence 10.7.2

$$\begin{array}{lllll} p \cong q & \text{iff} \quad p \xrightarrow{\tau} p' & \text{implies} \quad q \xrightarrow{\tau} \Longrightarrow q' & \text{and} & p' \approx q' \\ & \text{iff} \quad p \xrightarrow{\lambda} p' & \text{implies} \quad q \xRightarrow{\lambda} q' & \text{and} & p' \approx q' \\ & \text{(and vice versa)} & & & \end{array}$$

where $\cong$ is defined as follows:

$$p \cong q \text{ iff } p \approx q \ \wedge \ \forall r.\, p + r \approx q + r$$

A.5.0.5. CCS Axioms for Observational Congruence (Milner τ Laws)

$$\begin{array}{l} p + \tau.p = \tau.p \\ \mu.(p + \tau.q) = \mu.(p + \tau.q) + \mu.q \\ \mu.\tau.p = \mu.p \end{array}$$

A.5.0.6. CCS Dynamic Bisimulation 10.7.3

$$\begin{array}{lllll} p\ \Theta(R)\ q & \text{iff} \quad p \xrightarrow{\tau} p' & \text{implies} \quad q \xrightarrow{\tau} \Longrightarrow q' & \text{and} & p'\ R\ q' \\ & \text{iff} \quad p \xrightarrow{\lambda} p' & \text{implies} \quad q \xrightarrow{\lambda} \Longrightarrow q' & \text{and} & p'\ R\ q' \\ & \text{(and vice versa)} & & & \end{array}$$

A.5.0.7. CCS Axioms for Dynamic Bisimulation 10.7.3

$$\begin{array}{l} p + \tau p = \tau p \\ \mu(p + \tau q) = \mu(p + \tau q) + \mu q \end{array}$$

A.6. Temporal and Modal Logic

A.6.1. Hennessy - Milner Logic 10.5

The *satisfaction relation* $\models \subseteq P \times \mathscr{L}$ is defined as follows:

$$\begin{array}{lll} P \models \mathit{true} & & \\ P \models \neg F & \text{iff} & \text{not } P \models F \\ P \models \wedge_{i \in I} F_i & \text{iff} & P \models F_i \qquad \forall i \in I \\ P \models \Diamond_\mu F & \text{iff} & \exists p'.\ p \xrightarrow{\mu} p' \wedge p' \models F \end{array}$$

A.6.2. Linear Temporal Logic 11.1.1

We define the satisfaction operator $\models$ as follows:

- $S \models p$ if $0 \in S(p)$
- $S \models \neg\phi$ if it is not true that $S \models \phi$
- $S \models \phi_1 \wedge \phi_2$ if $S \models \phi_1$ and $S \models \phi_2$
- $S \models \phi_1 \vee \phi_2$ if $S \models \phi_1$ or $S \models \phi_2$
- $S \models O\phi$ if $S^1 \models \phi$
- $S \models F\phi$ if $\exists i \in \mathbb{N}$ such that $S^i \models \phi$
- $S \models G\phi$ if $\forall i \in \mathbb{N}$ it holds $S^i \models \phi$
- $S \models \phi_1 U \phi_2$ if $\exists i \in \mathbb{N}$ such that $S^i \models \phi_2$ and $\forall j < i\ S^j \models \phi_1$

A.6.3. Computation Tree Logic 11.1.2

Let (T, S, P) be a branching structure and $\pi = v_0, v_1, \ldots, v_n, \ldots$ be an infinite path. We define the $\models$ relation for as follows:

- $S, \pi \models p$ if $v_0 \in S(p)$
- $S, \pi \models \neg\phi$ if it is not true that $S, \pi \models \phi$
- $S, \pi \models \phi_1 \wedge \phi_2$ if $S, \pi \models \phi_1$ and $S, \pi \models \phi_2$
- $S, \pi \models \phi_1 \vee \phi_2$ if $S, \pi \models \phi_1$ or $S, \pi \models \phi_2$
- $S, \pi \models O\phi$ if $S, \pi^1 \models \phi$
- $S, \pi \models F\phi$ if $\exists i \in \omega$ such that $S, \pi^i \models \phi$
- $S, \pi \models G\phi$ if $\forall i \in \omega$ such that $S, \pi^i \models \phi$
- $S, \pi \models \phi_1 U \phi_2$ if $\exists i \in \omega$ such that $S, \pi^i \models \phi_2$ and for all $j < i$ $S, \pi^j \models \phi_1$
- $S, \pi \models E\phi$ if there exists $\pi_1 = v_0, v'_1, \ldots, v'_n, \ldots$ such that $S, \pi_1 \models \phi$
- $S, \pi \models A\phi$ if for all paths $\pi_1 = v_0, v'_1, \ldots, v'_n, \ldots$ we have $S, \pi_1 \models \phi$

This semantics apply both for CTL and CTL^*. The formulas of CTL are obtained by restricting CTL^*: a CTL^* formula is a CTL formula if the followings hold:

- A and E appear only immediately before a linear operator (i.e., F,G,U and O).
- each linear operator appears immediately after a quantifier (i.e., A and E).

A.7. μ-Calculus 11.2

We define the denotational semantics of μ-calculus which maps each predicate to the subset of states in which it holds as follows:

$$
\begin{aligned}
[\![x]\!]\rho &= \rho x \\
[\![p]\!]\rho &= \rho p \\
[\![\phi_1 \wedge \phi_2]\!]\rho &= [\![\phi_1]\!]\rho \cap [\![\phi_2]\!]\rho \\
[\![\phi_1 \vee \phi_2]\!]\rho &= [\![\phi_1]\!]\rho \cup [\![\phi_2]\!]\rho \\
[\![\neg\phi]\!]\rho &= V \setminus [\![\phi]\!]\rho \\
[\![true]\!]\rho &= V \\
[\![false]\!]\rho &= \varnothing \\
[\![\Diamond\phi]\!]\rho &= \{\, v \mid \exists v'. v \to v' \ \wedge\ v' \in [\![\phi]\!]\rho \,\} \\
[\![\Box\phi]\!]\rho &= \{\, v \mid \forall v'. v \to v' \Rightarrow v' \in [\![\phi]\!]\rho \,\} \\
[\![\mu x.\phi]\!]\rho &= \mathrm{fix}\, \lambda S.\, [\![\phi]\!]\rho[{}^{S}/_{x}] \\
[\![\nu x.\phi]\!]\rho &= \mathrm{Fix}\, \lambda S.\, [\![\phi]\!]\rho[{}^{S}/_{x}]
\end{aligned}
$$

A.8. π-calculus 12

A.8.1. π-calculys Syntax 12.1

$$
\begin{aligned}
p &::= \mathbf{nil} \mid \alpha.p \mid [x = y]p \mid p + p \mid p|p \mid (y)p \mid !p \\
\alpha &::= \tau \mid x(y) \mid \overline{x}y
\end{aligned}
$$

A.8.2. π-calculus Operational Semantics 12.2

$$\text{(Tau)}\ \frac{}{\tau.p \xrightarrow{\tau} p} \qquad \text{(Out)}\ \frac{}{\overline{x}\,y.p \xrightarrow{\overline{x}\,y} p} \qquad \text{(In)}\ \frac{}{x(y).p \xrightarrow{x(w)} p\{^w/_y\}}\ w \notin fn((y)p)$$

$$\text{(SumL)}\ \frac{p \xrightarrow{\alpha} p'}{p+q \xrightarrow{\alpha} p'} \qquad \text{(SumR)}\ \frac{q \xrightarrow{\alpha} q'}{p+q \xrightarrow{\alpha} q'}$$

$$\text{(Match)}\ \frac{p \xrightarrow{\alpha} p'}{[x=x]p \xrightarrow{\alpha} p'} \qquad \text{(ParL)}\ \frac{p \xrightarrow{\alpha} p'}{p|q \xrightarrow{\alpha} p'|q}\ bn(\alpha) \cap fn(q) = \varnothing \qquad \text{(ParR)}\ \frac{q \xrightarrow{\alpha} q'}{p|q \xrightarrow{\alpha} p|q'}\ bn(\alpha) \cap fn(p) = \varnothing$$

$$\text{(Com)}\ \frac{p \xrightarrow{\overline{x}\,z} p' \quad q \xrightarrow{x(y)} q'}{p|q \xrightarrow{\tau} p'|(q'\{^z/_y\})} \qquad \text{(Res)}\ \frac{p \xrightarrow{\alpha} p'}{(y)p \xrightarrow{\alpha} (y)p'}\ y \notin n(\alpha) \qquad \text{(Open)}\ \frac{p \xrightarrow{\overline{x}\,y} p'}{(y)p \xrightarrow{\overline{x}(w)} p'\{^w/_y\}}\ y \neq x \quad w \notin fn((y)p)$$

$$\text{(Close)}\ \frac{p \xrightarrow{\overline{x}\,(w)} p' \quad q \xrightarrow{x(w)} q'}{p|q \xrightarrow{\tau} (w)(p'|q')} \qquad \text{(Rep)}\ \frac{p|!p \xrightarrow{\alpha} p'}{!p \xrightarrow{\alpha} p'}$$

A.8.3. π-calculus Abstract Semantics 12.4

A.8.3.1. Strong Early Ground Bisimulation 12.4.1

$$p\ S\ q \Rightarrow \begin{cases} \text{if} \quad p \xrightarrow{\alpha} p' \text{ with } \alpha \neq x(y) \ \wedge\ bn(\alpha) \notin fn(q), \ \text{ then } \exists q'.\ q \xrightarrow{\alpha} q' \text{ and } p'\ S\ q' \\ \text{if} \quad p \xrightarrow{x(y)} p' \text{ with } y \notin fn(q), \text{ then } \forall w.\ \exists q'.\ q \xrightarrow{x(y)} q' \text{ and } p'\{^w/_y\}\ S\ q'\{^w/_y\} \end{cases}$$

We define the strong early ground bisimilarity as follows:

$$p \overset{\circ}{\sim}_E q \Leftrightarrow p\ S\ q \text{ for some strong early ground bisimulation } S$$

A.8.3.2. Strong Late Ground Bisimulation 12.4.2

$$p\ S\ q \Rightarrow \begin{cases} \text{if} \quad p \xrightarrow{\alpha} p' \text{ with } \alpha \neq x(y) \ \wedge\ bn(\alpha) \notin fn(q), \ \text{ then } \exists q'.\ q \xrightarrow{\alpha} q' \text{ and } p'\ S\ q' \\ \text{if} \quad p \xrightarrow{x(y)} p' \text{ with } y \notin fn(q), \ \text{ then } \exists q'.\ \forall w.\ q \xrightarrow{x(y)} q' \text{ and } p'\{^w/_y\}\ S\ q'\{^w/_y\} \end{cases}$$

We define the strong late ground bisimilarity as follows:

$$p \overset{\circ}{\sim}_L q \Leftrightarrow p\ S\ q \text{ for some late early ground bisimulation } S$$

A.8.3.3. Strong Early Full Bisimilarity 12.4.3

$$p \sim_C q \Leftrightarrow p\sigma \overset{\circ}{\sim}_E q\sigma \ \text{ for every substitution } \sigma$$

A.8.3.4. Strong Late Full Bisimilarity 12.4.3

$$p \sim_C q \Leftrightarrow p\sigma \overset{\circ}{\sim}_L q\sigma \ \text{ for every substitution } \sigma$$

A.9. PEPA 15

A.9.1. PEPA Syntax 15.2.1

$$P ::= (\alpha, r).P \mid P + P \mid P \underset{L}{\bowtie} P \mid P/L \mid C$$

A.9.2. PEPA Operational Semantics 15.2.2

$$\frac{}{(\alpha, r).P \xrightarrow{(\alpha,r)} P} \qquad \frac{P \xrightarrow{(\alpha,r)} P'}{P + Q \xrightarrow{(\alpha,r)} P'} \qquad \frac{Q \xrightarrow{(\alpha,r)} Q'}{P + Q \xrightarrow{(\alpha,r)} Q'}$$

$$\frac{P \xrightarrow{(\alpha,r)} P' \quad \alpha \notin L}{P/L \xrightarrow{(\alpha,r)} P'/L} \qquad \frac{P \xrightarrow{(\alpha,r)} P' \quad \alpha \in L}{P/L \xrightarrow{(\tau,r)} P'/L}$$

$$\frac{P \xrightarrow{(\alpha,r)} P' \quad \alpha \notin L}{P \underset{L}{\bowtie} Q \xrightarrow{(\alpha,r)} P' \underset{L}{\bowtie} Q} \qquad \frac{Q \xrightarrow{(\alpha,r)} Q' \quad \alpha \notin L}{P \underset{L}{\bowtie} Q \xrightarrow{(\alpha,r)} P \underset{L}{\bowtie} Q'}$$

$$\frac{P \xrightarrow{(\alpha,r_1)} P' \quad Q \xrightarrow{(\alpha,r_2)} Q' \quad \alpha \in L}{P \underset{L}{\bowtie} Q \xrightarrow{(\alpha,r)} P' \underset{L}{\bowtie} Q'} \quad \text{where } r = \min(r_\alpha(P), r_\alpha(Q)) * \frac{r_1}{r_\alpha(P)} * \frac{r_2}{r_\alpha(Q)}$$

$$\frac{P \xrightarrow{(\alpha,r)} P' \quad (C \stackrel{\text{def}}{=} P) \in \Delta}{C \xrightarrow{(\alpha,r)} P'}$$

A.10. LTL for Action, Non-determinism and Probability

Let S be a set of states, T be a set of transitions, L be a set of labels, $D(S)$ and $D(L \times S)$ be respectively the set of discrete probabilistic distributions over S and over $L \times S$.

- CCS: $\alpha : S \longrightarrow \mathscr{P}(L \times S)$
- DTMC: $\alpha : S \longrightarrow (D(S) + 1)$
- CTMC: $\alpha : S \to S \to \mathbb{R}$
- Reactive Markov Chain: $\alpha : S \to L \to (D(S) + 1)$
- Generative Markov Chain: $\alpha : S \to (D(L \times S) + 1)$
- Segala Automata: $\alpha : S \to \mathscr{P}(D(L \times S))$
- Simple Segala Automata: $\alpha : S \to \mathscr{P}(L \times D(S))$

A.11. Real-valued Modal Logic

A.12. Larsen-Skou Logic 14.1.1.1

We define the Larsen-Skou satisfaction relation as follows:

$$\begin{array}{lcl} s \models \textbf{true} & & \\ s \models \varphi_1 \wedge \varphi_2 & \Leftrightarrow & s \models \varphi_1 \text{ and } s \models \varphi_2 \\ s \models \neg\varphi & \Leftrightarrow & \neg s \models \varphi \\ s \models \langle l \rangle_q \varphi & \Leftrightarrow & \alpha\, s\, l\, [\![\varphi]\!] \geq q \text{ where } [\![\varphi]\!] = \{s \in S \,|\, s \models \varphi\} \end{array}$$