1 Introduction

1.1 Modeling Distributed Systems

The theory of distributed computation can be studied profitably by means of a class of models that can represent distribution as a first-class concept. Among the many semantic models proposed in the literature with this aim, *Petri nets* – so-called after the name of their inventor Carl Adam Petri [93] – constitute a pivotal semantic model for a number of reasons, including at least the following:

(i) Distribution is indeed a first-class concept – which is not the case for, e.g., *labeled transition systems* [76, 38, 105, 45], a model commonly used for giving semantics to process algebras [84, 8, 1, 5, 106] in an *interleaving* style, where the execution of two independent parallel actions is modeled in the same way as their causal sequential execution in either order.

(ii) Since Petri nets possess a notion of (distributed) state, they can be used to model recursive behavior with a finite structure – which is not the case for other models of concurrency such as *event structures* [86, 114].

(iii) Petri nets are a widely studied semantic model (see, e.g., [92, 101, 28, 46] and the references therein), equipped with precise and simple behavioral semantics, as we will see in the following chapters of this book.

(iv) Interesting analysis techniques, which are decidable in some cases, are available for Petri nets (see, e.g., [92, 99, 46] and the references therein); these techniques are sometimes supported by automatic or semi-automatic software tools (see, e.g., [108, 109] for surveys on Petri net tools).

(v) And, finally, in the literature there is a large number of case studies in which Petri nets are applied to the modeling, analysis and verification of real distributed systems (see, e.g., [102, 100] and the references therein).

In contrast to labeled transition systems, whose states are monolithic entities, Petri nets describe the global state of a system as composed of a collection of local states. A transition does not involve the entire global state; rather it applies only to some local states. According to the usual Petri net terminology, a local state s is called a *place*, while a global state m, called a *marking*, is a multiset of local states; hence, since m is a multiset, there may be more than one instance of a certain local state s in m; a net *transition* is connected to all the places that take part in the local operation that this transition wants to model. Graphically, a place is represented by a small circle; an instance of a local state s, called a *token*, by a little bullet inside place s; and a net transition by a small box, which is connected to the places from which it removes tokens and to the places into which it produces tokens.

Petri nets are often used to model production systems, where a place is seen as a resource type, the number of tokens on such a place denotes the number of instances of resources of that type and a transition is used to represent an activity that consumes and produces resources. However, in this book, since we use Petri nets as a tool for studying the theory of distributed computation, the interpretation of places and transitions follows a different intuition: A place represents a sequential process type, each token on such a place denotes an instance of a process of that type and a transition represents the evolution of one or more sequential processes, which possibly interact by synchronizing.

In the literature several different classes of Petri nets have been proposed for different purposes. Some simple net models include *Condition/Event systems* [99] and *Elementary net systems* [103], where it is required that places can hold one token at most; or *Free choice nets* [26], where a net of this sort must possess a particular structure. Other, more advanced models include, e.g., *Nets with inhibitor arcs* [92, 70, 18], where transitions have the capability to test whether a place contains zero tokens.

Among these many variants, we focus our attention on the following three classes of Petri nets, which are important as they precisely characterize three important classes of computational systems:

- *Linear nets* (also called *finite-state machines*). The transitions have a specific, very restricted shape (they consume one token and produce one token at most), and the initial marking is a singleton, so that all the reachable markings are singletons. Therefore, this class models *sequential systems* because there is only one active sequential process (the unique token around in the linear net).
- *Forking nets* (also called *BPP nets*). The transitions consume one single token but may produce many tokens; moreover, the initial marking can be any multiset of local states. Therefore, this class models *noncommunicating parallel systems* because the initial marking specifies the collection of initial sequential processes composing the distributed system, and these processes

cannot communicate (as the transitions consume only one token) but can spawn other processes (as the transitions may produce multiple tokens).

- *Petri nets* (*tout court*, also called *Place/Transition Petri nets*). The transition may consume multiple tokens and may produce multiple tokens. Therefore, this class models *communicating parallel systems* because a transition consuming multiple tokens models a synchronization among the sequential processes represented by those tokens.

1.2 Behavioral Equivalences

A net transition is labeled by an action taken from a given alphabet, which represents its observable content. Hence, the label of net transitions (and, possibly, also the structure of such transitions) can be used to compare the behavior of different nets. So a natural question arises: When are two nets to be considered behaviorally equivalent? The answer to this question is not unique: We will see in Chapter 5 a rich panorama of the many different equivalences that have been defined for Petri nets. Nonetheless, we have made a precise choice by proposing for each class of nets a behavioral equivalence such that it is

(i) *resource sensitive*, meaning that it relates markings of the same size only;
(ii) *causality respecting*, meaning that it captures correctly the causal dependencies among the performed events; and, finally,
(iii) *decidable*, i.e., the equivalence-checking problem can be solved algorithmically.

1.2.1 Resource Sensitive

Starting from linear nets, we advocate the use of *bisimulation* [90, 84] as the suitable behavioral relation for these nets. However, our definition slightly deviates from the previous definition in the literature (originally defined on labeled transition systems) in that it does not consider the empty marking behaviorally equivalent to a deadlock place, i.e., *it distinguishes successful termination from deadlock*. This difference is not irrelevant: As a token denotes a sequential process that is using a processor, if a transition produces a token in a deadlock place, then that sequential process is stuck but still uses the processor; on the contrary, if a transition produces no token at all (i.e., it reaches the empty marking), then that sequential process has terminated its execution successfully, so that the processor it was using is now freely available for another sequential process. From this perspective, we could say that our variant definition of bisimulation is *resource sensitive*, where the resource we are considering is the processor.

When considering forking nets, we generalize the definition of bisimulation to *team bisimulation* [50, 54], which is still resource sensitive, as it relates markings of the same size only, so that team bisimulation equivalent noncommunicating parallel processes use the same amount of computational resources.

When considering Petri nets, we introduce two behavioral equivalences that generalize team bisimulation, namely *place bisimulation* [3] and *structure-preserving bisimulation* [39], both resource sensitive. The main reason for proposing the former is its simplicity and the fact that the behavioral equivalence it induces is decidable for Petri net, while the latter has better mathematical properties.

Summing up, the first main motivation for our selected equivalences is that they are resource sensitive.

1.2.2 Time + Space ⇒ Causality

Another common feature of all the selected behavioral equivalences is that they respect causality.

For linear nets, this is quite obvious: As there is only one token around, the temporal order of action execution implies the causal order of these actions. In fact, if a sequential process performs a sequence ab, i.e., two transitions with labels a and b respectively, then the token consumed by the b-labeled transition is precisely the one produced by the a-labeled transition. In other words, the causality relation among the performed actions is directly determined by the temporal order in which the token is first produced and then consumed. Hence, the definition of bisimulation on a linear net is causality respecting.

For forking nets, this is less obvious but still easily understandable. Since a transition consumes only one token, it is enough to take care of the flow of token production and consumption: If one transition t consumes one token produced by the previous execution of some transition t', then the only immediate cause of t is exactly t'. To take into account this causality flow, team bisimulation requires that if two places are related, then the matching transitions of the bisimulation game should be such that the reached markings are bijectively related by the team bisimulation relation itself. This implies that the threads of computations that the independent tokens produce are to be bijectively related by means of a team bisimulation on the net places. Therefore, the information about the bijection between the two distributed states (i.e., on the current markings of interest) and the time ordering in the corresponding computations they perform are enough to determine that the actual causal order among the performed events in these two computations is the same.

For Petri nets, the situation becomes more complex, as the immediate causes of a transition can be a whole multiset of transitions performed previously in time. However, rather surprisingly, place bisimulation and structure-preserving bisimulation, which are both smooth generalizations of the team bisimulation idea, respect causality, since we will formally prove in Chapter 5 that the very complex, but largely agreed upon, definitions of causality-respecting behavioral equivalences (e.g., *causal-net bisimulation* [39, 54] or *fully-concurrent bisimulation* [10]) are closely related to the much simpler ones mentioned above. In fact, one of the goals of this book is to show that causality-respecting behavioral equivalences can be defined in a rather simple manner over the three classes of nets we consider.

1.2.3 Why Causality?

One may wonder why causality should be an important feature of a behavioral equivalence. Some reasons for this choice include at least the following:

- *Error recovery*. If a bug occurs during a computation, it may not be obvious to discover what its causes are if the semantics does not make causality observable. Conversely, if causality is modeled, then one can focus attention only on the subcomputation that caused the occurrence of the bug, hence making the fixing much more efficient.
- *Performance*. If we assume that transitions with the same label take the same amount of execution time for completion, and if we also assume that each transition is performed as soon as possible, then two behaviorally equivalent markings perform their tasks in exactly the same execution time if the behavioral equivalence is causality respecting.
- *Verification*. The causality-respecting behavioral equivalences we advocate for forking nets and Petri nets possess better verification algorithms than those based on so-called *interleaving* (cf. Chapter 5 for an overview of interleaving behavioral equivalences). In particular, while interleaving bisimulation is undecidable [68] for Petri nets, place bisimulation equivalence is decidable [52].

1.3 Modal Logic

Once the bisimulation-based behavioral equivalence for the class of Petri nets of interest has been defined, it is interesting to investigate whether it possesses a modal logic characterization, i.e., whether there exists a modal logic such that two markings are behaviorally equivalent if and only if they satisfy the same modal logic formulae. The answer to this question is positive for each of the three classes of nets.

For linear nets, bisimulation equivalence is characterized logically by a simple modal logic, called *bisimulation modal logic* (BML), resembling *Hennessy–Milner logic* [62] (HML) but with the ability to distinguish successful termination from deadlock. For forking nets, team bisimulation equivalence is characterized logically by *team modal logic* [50] (TML), a suitable extension of BML with the introduction of an operator of parallel composition of formulae. For Petri nets, structure-preserving bisimulation equivalence is characterized logically in a rather complex manner: First, with *marking logic* (MKL) we can define formulae that may contain the actual names of places, so that different markings cannot satisfy the same formulae; then, we define a corresponding logic, *linking logic* (LKL), such that structure-preserving bisimulation equivalent markings satisfy not exactly the same formulae but rather corresponding formulae that are structurally the same but that may differ in the actual choice of the concrete names of places.

1.4 Process Algebra

The next step is to define a suitable process algebra representing the models of interest. In fact, for each one of the three classes of nets, we define a corresponding process algebra such that:

- each term p of the process algebra is given a semantics in terms of a net $Net(p)$ of that class (*only* nets of that class can be modeled by the process algebra); moreover,
- for each net $N(m_0)$ of that class (i.e., for each net N with initial marking m_0), we can single out a term p of the corresponding process algebra, whose semantics is a net isomorphic to $N(m_0)$ (*all* the nets of that class are represented by the process algebra, up to net isomorphism); and, finally,
- all the operators of the process algebra are necessary to get these results (no superfluous operator is introduced).

Therefore, since we argued that each class of nets is meant as a suitable class of computational systems, our contribution amounts to *alphabetizing* these three classes of systems, i.e., providing three process algebras, as simple as possible, each one representing all and only the computational systems of a certain class.

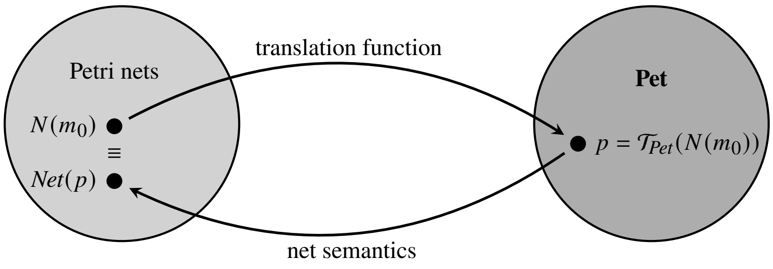

Figure 1.1 Graphical description of the representability theorem for Petri nets and the process algebra **Pet**.

For linear nets, the corresponding process algebra is called **Lin**, and it is essentially a variant of **finite-state CCS** [84, 45], whose operators are the deadlock process **0**, a family of simple processes of the form $a.\mathbf{1}$ that can perform a and then successfully terminate, action prefixing $a.p$, choice $p + q$, and process constants $C, D, \ldots$, equipped with a definition useful for describing recursive behavior.

For forking nets, the corresponding process algebra, called **Fork**, is similar to **BPP** [21] and enriches **Lin** with the binary operator of *asynchronous* (i.e., without synchronization capabilities) parallel composition $p \mid q$; in particular, in a term $a.p$, the action a may prefix not only a sequential process (as for **Lin**) but also a parallel process.

For Petri nets, the corresponding process algebra is called **Pet** and extends **Fork** by enhancing the prefix operator to include atomic sequences of inputs (besides single inputs and single outputs). It also enhances its parallel composition operator to allow for communication (according to a generalization to sequences of the binary, handshake, synchronization discipline of **CCS** [84, 45]) and by also including the **CCS** restriction operator $(\nu a)p$ (to force synchronization within p), to be used at the top level only. The combination of input sequence prefix and parallel composition allows a mechanism for multi-party synchronization to be implemented, which is modeled as an atomic sequence of binary, **CCS**-like synchronizations.

Figure 1.1 describes graphically one of the three *representability theorems* in this book, namely that for the case of Petri nets and the process algebra **Pet**: Given a marked Petri net $N(m_0)$, the *translation function* $\mathcal{T}_{Pet}(-)$ maps $N(m_0)$ to the **Pet** process term $p = \mathcal{T}_{Pet}(N(m_0))$; then, the net semantics maps the term p to the Petri net $Net(p)$, such that $Net(p) \equiv N(m_0)$, where $\equiv$ denotes net isomorphism equivalence.

1.5 Compositionality and Congruence

The first great advantage of having a suitable process algebra representing the models of a certain class of nets, up to isomorphism, is that it provides *linguistic support* for this class: A succinct, textual (i.e., linear), implicit representation of the net model as a term in the process algebra corresponding to the class of net models. A case study illustrating this fact is described in Section 4.6.4.

Another good reason for developing a process algebra that can describe all the nets of a certain class is that it allows for *compositional modeling and reasoning*. Whenever a complex distributed system is to be modeled by a net, it is often convenient to identify its sequential subcomponents, to model each of these separately, to reason on each of them independently of the other subcomponents, and then, possibly, to derive global properties of the whole distributed system from the local properties of the sequential subcomponents.

As an instance of the kind of analyses that are possible by *compositionality*, suppose a net system is described by the process algebraic term $p_1 \mid p_2$, denoting two processes p_1 and p_2 composed in parallel. It often happens that the composite system $p_1 \mid p_2$ satisfies a certain property if this property holds for the constituents p_1 and p_2. As an instance of a property of this form, consider "*the system always terminates its computation*." Hence, instead of checking this property against the large global state space of the composite system, we can just check if this holds for the two much smaller local state spaces of the two constituents.

This compositionality reasoning is even more effective when considering behavioral equivalences that are *congruences* with respect to the operators of the process algebra of interest. For instance, when checking whether two composite systems, say $p_1 \mid p_2$ and $q_1 \mid q_2$, are behaviorally equivalent, it is much more convenient to check separately whether, e.g., p_1 is behaviorally equivalent to q_1, as well as whether p_2 is behaviorally equivalent to q_2, instead of checking this on the two much larger global state spaces of $p_1 \mid p_2$ and $q_1 \mid q_2$, because when the equivalence is preserved by the operator of parallel composition (i.e., it is a *congruence with respect to parallel composition*), then these two local checks are sufficient to ensure that $p_1 \mid p_2$ and $q_1 \mid q_2$ are behaviorally equivalent, too. A case study illustrating the advantage of compositional reasoning by congruence is described in Section 4.7.3.

Interestingly enough, all the equivalences developed for the classes of nets considered in this book are congruences for the operators of the corresponding process algebra, so that for each class of nets we have developed a fully satisfactory *compositional semantics* over the terms of the corresponding process algebra.

1.6 Algebraic Properties and Axiomatization

One further good reason for developing a process calculus to describe a class of nets is that it allows for the definition of equational theories (usually called *axiomatizations*) characterizing the behavioral congruences of interest. In fact, by means of an axiomatization, it is possible to offer an alternative, completely syntactic, technique to prove when two nets are behaviorally congruent.

More precisely, so far, two nets N_1 and N_2 can be proved behaviorally congruent by checking whether there exists a suitable behavioral relation over them; but now, as these two nets are actually represented by suitable terms of the same process algebra, say p_1 and p_2, the proof that N_1 and N_2 are behaviorally congruent can alternatively be done with a purely syntactical argument, by showing that the term p_1 can be equated with the term p_2 by means of some equational deductive proof.

To do so, first we have to study which algebraic properties that congruence satisfies, and, based on these, we can possibly define a *sound and complete* axiomatization characterizing the behavioral congruence syntactically.

Interestingly enough, all the behavioral equivalences (which are also congruences) developed for the process algebras considered in this book have been axiomatized (i.e., they have been characterized by a finite set of axioms), even if, in the case of Petri nets and the corresponding process algebra **Pet**, this axiomatization is not complete.

1.7 Structure of the Book

The body of the book consists of six chapters. After this introduction, the next three all have the same structure and are organized as follows:

(i) First, the class of nets is introduced (linear nets in Chapter 2, forking nets in Chapter 3, Petri nets in Chapter 4).
(ii) Then, the behavioral equivalences of interest are presented (bisimulation equivalence in Chapter 2, team bisimulation equivalence in Chapter 3, place bisimulation and structure-preserving bisimulation equivalences in Chapter 4).
(iii) A modal logic characterization of these equivalences is then described (BML in Chapter 2, TML in Chapter 3, MKL and LKL in Chapter 4).
(iv) A suitable process algebra for the corresponding class of nets is introduced (**Lin** in Chapter 2, **Fork** in Chapter 3, **Pet** in Chapter 4).
(v) A verification that the behavioral equivalences of interest are congruences is also provided, together with a study of their algebraic properties.
(vi) Finally, an axiomatization of the behavioral congruence is proposed.

Chapter 5 surveys some of the many different behavioral equivalences that have been proposed in the literature for Petri nets. It also gives the proof that place bisimulation equivalence and structure-preserving bisimulation equivalence are both causality-respecting behavioral relations. Finally, Chapter 6 hints at possible extensions of the theory presented in this book.