Mobility and Computation

U. Ferreira
Escola Politécnica, R. Caetano Moura, Federação, Salvador, Brazil
e-mail: *u.ferreira@philosophers-fcs.org*

Abstract

Mobile agents and the Internet have brought new ideas to theoretical foundations of computer science in the last few years.

As an example, in 1999, I had an interesting conceptual insight over computation: "...at the moment that I conceive the idea of moving computation from one place to another, I also observe that a general notion of computation transcends pure mathematics and meets the physical world". This itself requires new, informal and philosophical discussions in the foundations of computer science. Later, in 2001, "and because the universe is on the move, computation is essentially mobile."

The present paper discusses some meaning of computation, provides a different semantics and present a formalized, physical and abstract model after my simplification. The present model makes use of four forms of mobility, namely strong mobility, intentional unity mobility, non-intentional unity mobility, and broadcast mobility. In this paper, I present other arguments for the most general and unified notion of computation, although it is only one among other good proposals. Mobility and global computing form two different classes of argument.

1 Introduction

It is not easy to perceive the characteristics of the era in which we all are, because an era transcends the life time of human beings as we miss comparisons which are more realistic than what the literature can teach. The Internet has grown very quickly. This global infrastructure, along with other recent technologies, such as satellite television, mobile phones and portable computers, have not only changed humans' behavior but have also made this planet psychologically smaller than ever. On the one hand, this new apparatus has led to new terminology regarding mobility[17], computation[69, 80], programming languages[57], distributed systems[7, 13] and mobile agents, in such a way that this terminology deserves care, regarding the appropriateness of its use. On the other hand, I observe that one notion of computation cannot be captured using mathematics. As an example, one may regard parallel and concurrent computing, some forms of mobility, side-effects, unreliability of the physical media and other factors as non-mathematical, or mathematical via physics, although they can be abstractly captured by algebra and categories up to some extent.

To exploit this view in this paper, I consider the semantics of computation as based on its physical nature, with some rough simplifications in the present model as I do not deeply investigate psychological issues. It is also part of the present view to regard the traditional theory of computation, which is based on recursive functions, as extremely and historically important although, perhaps from the present standpoint, that theory does not capture what I am calling computation here. In [58], the authors describe a method for proving termination of recursively defined functions based on ordinal measure, and such contributions are very relevant, even adopting a non-mathematical perspective, or a different philosophical view.

In this paper, I use interchangeably the terms "computer science" and "computing science" as meaning the same science. In the thirties of the last century, "computer" was the person who used to calculate, normally some woman. Although I have my own personal view, in the present paper, I do not discuss the philosophical issue of whether humans are machines, nor the issue on whether God exists. However, such questions are probably what distinguish what I refer to as *the fourth level of computation* from a possible fifth level, and also what distinguish this possible fifth level from higher levels (until the level of God). This may be seen as a kind of different levels of the Church-Turing thesis stating that there is a unique level of computation, or, alternatively, as different levels of negations of the latter philosophical idea.

Some questions arise that might conceptually interest the theory of computer science. For example, let us imagine two mobile agents that travel in space at the same speed. The first one is traveling indefinitely. The second one is running a simple algorithm that, when it has met the first, halts. Should one regard the second agent as running an infinite computation? If we adopt the point of view that computation is a physical[50] concept and that space is flat, the answer is *yes*.

Another issue is whether mobility introduces new elements in the theory of computing science or not. A number of academic and theoretical work, to some of which I make references in the present paper, forms strong evidence that the computing science community agrees it does. Under certain philosophical perspective, we can also observe that mobility is a primitive in computation and,

since this, I provide a formal and physical semantics of computation (one of the proofs that the present meaning is more general is by introducing one different form of code mobility which can be via broadcasting media, such as radio or satellite television) and, finally, I mention other factors of the real world, such as faults and delays in network connections, as relevant to the probably more general and physical notion of computation. For the formal semantics, I write the rules using a space-time logic that I briefly introduce here. In this paper I argue that both physical and philosophical factors are part of the broader notion. Indeed, theory of computation has strong connection with philosophy, and this claim extends the connections between logic[22, 46] and philosophy[24]. Here, I simply discuss the matter while I present a number of examples that have been selected during these years of research and reflexions.

Regarding the physical nature of computation, I introduce an operational semantics of computation that includes space and time, as well as captures the present four forms of mobility. For the present PhD thesis, these factors are enough to demonstrate that, although pure mathematics and logics are still going to be used as computation and to simulate reasoning[53], they do not capture more general notions of computation. In [61], there is an interesting introduction on mobile agents, where the author shows evidence of benefits which they achieve. For more advanced literature on this subject, [65], for instance, among some others.

As regards philosophical factors, one enters a subjective, informal and possibly psychological world, the real world, transcending the mathematical and logical language as well as traditional computer science texts. The notion of computation should not be confined to a unique and universal concept, but instead, there should be diversity, e.g. the concept of computation depends on the defined underlying machine, although these machines share common properties. Further, for each different notion of computation, there can be different theories of computation, including different theories of computability. Further, the referred to term *theory*, for instance, starts having a broader meaning, which not only includes the traditional one, from deductive logics, but also philosophical theories. Machines are becoming gradually more complex. A global computer, for instance, is an abstract and general machine atop some internet services and includes the notion of code mobility. There are other approaches to global computation, such as [23]. On the one hand, such a computer provides a more general notion in comparison to those notions defined since Alan Turing and others, who did not consider mobility as a physical primitive. On the other hand, the geographical distribution introduces other factors to the definition of the global computer that cannot be neglected. Because every computer has its repertoire of operations which not only defines but also constraints the capability of the machine, any global computer has to provide a repertoire of operations that depends on ethics, common sense and the laws of the civilized world, e.g. some issues are discussed in chapters of [52]. Those factors belong to the real world, not to an idealized world such as that of mathematics. As an example, a state or country can establish some law to prevent unsolicited e-mail, say, messages of advertisement have to contain the string "Advert" in the beginning of their subject fields. One of the subjective parts of this is that the laws for e-mail often apply to the receiver and not to the sender, or to the sender and not to the receiver, and this makes it more controversial. At a different level, the same holds for mobile agent systems, which are typically spread out among

different cultures. On the one hand, global computers have to establish what can be computed. On the other hand, laws of behavior depend on time and space, among other factors. As another hypothetical example, suppose that a country has a law that adopts the policy to use only software open to its public administration. However, should a global computer guarantee privacy of mobile agents? Can the country impose such a constraint on incoming code? Such a discussion might be controversial.

Therefore, not only in the present paper, both formal work and informal discussions are significant. Informal discussions precede formalizations.

Section 2 is somewhat ontological. I discuss terminologies of notions related to agents[21] or mobility. Section 3 is very conceptual as I introduce a view of mobility detached from other concepts and I discuss its relationships with other related concepts. Section 4 discusses other complementary notions such as distribution and centralization. Section 5 introduces an intuitive notion of computation, and in section 6 I present an algorithm that solves the halting problem under the assumption of a finite-tape Turing machine. In section 7, I provide one notion of computation that is somehow broader than the well-established notions[71], such as Turing machines, $\lambda-$ and $\pi-$ calculi. At a higher level, the proposed notion is based on physical, mental and philosophical factors, besides mathematics[30]. I see computing science as a table whose legs are these four studies. In this way, I extend the conventional operational semantics by adding space, time and mobility, as well as defining states in a more sophisticated way, in comparison to the basic literature. Finally, section 8 concludes the paper.

2 Agents

In this paper, I consider four forms of mobility. I dedicate this section to agents.

The term *agent* has been used by both the AI[41] and distributed systems (DS) communities with different meanings and at different levels[34]. In addition, the term *agent* in English has the same spelling as in French, and has almost the same spelling and meaning as the term *agente* in Italian, Spanish and Portuguese. In these languages, *agent* or *agente* can normally mean "a person who acts on the behalf of another person or other people" or "a person who does something or causes something to happen". However, the word *agente*, comes from the verb *ago* in Latin, which means *to act*.

The term *mobile agent* is somewhat ambiguous. For instance, robots are agents that act physically on the environment, some of them are mobile and they have been referred to (by some) as mobile agents, but they are objects very different from mobile agents with which some researchers in the programming languages and distributed systems communities deal. Researchers from AI have commonly used the term *software agent* to differ from the other forms of agency. In this paper, I use the term mobile agent in the context of code mobility, rather than robotics. As well as code mobility, one of the four forms of mobility will be roughly related to the latter meaning of agent, from robotics.

In both fields of computer science, DS and AI, it has been noted that the term *agent* still lacks a clear and standard definition[34]. An interesting question now is whether it is really necessary to have a clear and standard definition of the term agent by a few particular computer scientists, or whether to let

the problem of these notions and terminology disappear naturally in future work. That is, if different communities have used the same word in English with different meanings, it might also be the case that both technologies and fields have things in common. From this perspective, we ought to explore and investigate this combination.

Since old times, societies have developed and the term agent has become more sophisticated. Nowadays, travel agents represent passengers in transactions with airlines, for instance. Agents are able to *act* on users' behalf and, because of this, must have autonomy and authorization to do so. In such a more complex context, intelligence is one of the desirable requirements for human, hardware or software agents. This is one of the points where AI has much to contribute. More than this, there is a subtle emerging area of research related to agent technology: users not only want agents to be as intelligent as themselves, but also want agents to behave in accordance with their corresponding psychological profiles. What does psychological profile mean? how to program it? Answers for such philosophical questions have to be established before implementation.

I believe that, although the meanings in the terminologies of agents from DS and AI are different and apply to different levels, they can easily complement each other. Thus, programming languages can support this integration.

3 Mobility and some related concepts

This section is a conceptual discussion on the foundations of computing science in the presence of mobility. There are good surveys on code mobility such as [27, 34, 61]. There are other important contributions. Briefly, from the technological standpoint, code mobility came from a refinement of the client-server paradigm of distributed systems. The well-know paradigms for code mobility are: remote evaluation, code on demand and mobile agents. Additionally, there have existed two forms of code mobility: week and strong. In the present paper, I am particularly interested in strong mobility, which requires the implementation by the mobile agents paradigm, although there are mobile agents systems that provide a weak form of mobility.

On the one hand, mobile agents technology was initially developed to solve or minimize technical problems, in particular in a distributed environment where performance is regarded as important. Furthermore, the Internet is a shared resource, users want to share their resources in a controlled way, and this technological scenario contributes to development of mobile agents technologies. Thus, transactions usually need several messages between partners and, when this case holds, they ought not to be performed remotely but mainly by local communication[68, 74] between a mobile agent and another agent. This requires new programming languages concepts and constructs, and some have been designed considering agent migration as priority, such as [26, 60].

As already written, the present notion of computation provides four forms of mobility. In addition to code mobility, the movements of a robot in a corridor, and the movements of a portable computer running a program in a transport on the move, at least for computer science, should *not* be examples of the same form of mobility for, although the computation moves as a consequence of hardware mobility, robots move *intentionally*, in contrast with portable computers,

although some technologies can permit both implementations to be *aware* of physical positions of the underlying physical machine.

"*Mobile computation*"[16] has been used informally to mean the computation supported by mobile code applications. The longer the distance the stronger the argument towards mobile code systems. However, although electronic commerce, for example, can be conceived without code mobility, *CPU loan or rental* is a kind of application impossible to be done *on-line* on a computer that does not provide code mobility. Mobile agents typically come and use someone's CPU, or even many CPUs in parallel. Although very simple, this example is evidence that code mobility provides a physical and probably more general meaning of what can be computed in the real world, and not simply a new technology.

Conceptually, one can observe that *mobility* has always been a basic concept in computer science, as well as part of some models of computation. For example, although there are books that provide a symbolic definition of a Turing machine such as [66], a well-accepted metaphor since Turing himself is based on one head that *moves* along a tape as a result from the current state and the transition function.

There are examples in other models. In a Petri net the control also *moves* in a similar way to a finite-state machine, either deterministic or not. The β-reduction rule in λ-calculus is also a move in a sense. Furthermore, any von Neumann machine, any sequential computer, provides mobility at several levels: a variable assignment is a move and a copy. The digital and analogical circuits also move bits and electrons, and so on. In order to generalize, every bit that moves from one part of the computer to another may be conceived as the simplest form of code mobility: it has source, content and destination. Because these models of computation provide some form of simple mobility, code mobility should be a *primitive* of a more general notion of computation. The fact that some mobile agent can simulate a Turing machine write/read head or a token in a Petri net is only one example of the generality of the notion of mobility.

Summarizing the forms of mobility dealt with in the present paper, I itemize four different ones:

- Strong mobility: the destination is stated explicitly and hardware or computational environment (CE) does not move (existential software movement in a sense).
- Broadcast mobility, **SpreadOut** primitive: the destination is not explicit and hardware or CE does not move (universal software movement in comparison to the strong mobility).
- Non-intentional hardware mobility: the hardware moves (or it is moved) and the software might be aware or not.
- Intentional unite mobility, **wemove** command: it is an intentional form of mobility of the computational environment (CE), in a sense. Robots movements and people walking in the streets, might also be seen as particular case here.

There are other forms of mobility.

4 Other Concepts

In [4], the author presents a model of distributed computation which is based on a fragment of π-calculus relying on asynchronous point-to-point communication. The same author then enriches the model with some features. In general, nowadays, many researchers who work in the code-mobility community are in some distributed systems group, and most call for papers of conferences on distributed systems includes mobile agents technologies. The connection between the two notions is indeed very strong.

Here, an important issue is: are mobile agent systems distributed systems? If one thinks carefully, the answer may be *no*. Research on mobile-code technology includes research on programming languages, design and implementation, and such languages make programmers be aware of resources, which contrasts with the philosophy of distributed systems in its traditional sense.

But, returning to the primary and conceptual level, the terms centralization and distribution are often related, and they can oppose or complement. Although we might prefer to use distributed computing for technical reasons such as efficiency, robustness and security, the concept of centralization is necessary even in computing. Taking the human body as a metaphor, the human circulatory system consists of *one* heart, veins spread out in the body and blood. Some of us know that the movement of blood in veins towards the heart represents centralization while the opposite movement away from the heart to the parts of the body represents distribution. This is a natural example showing that centralization and distribution can be mutually beneficial, and even be necessary for each other. Here, mobility is a third important component, represented by the movement of blood in both directions.

Similarly, although complex systems are normally distributed, *mobile code systems are not distributed systems* but they are related[10, 77]. Likewise, *mobility is not distribution*, but these concepts can coexist. These concepts exist at different levels of abstraction. For example, one can implement a distributed system using code mobility.

The concept of centralization is present in mobile code systems. For example, the concept of centralization applies to the level of programming. In this case, a central component is the programming language, as it provides standards and imposes constraints to the whole system. As another example, agents can move to a central place, a specific interpreter, and communicate with each other locally. This independence from centralization and distribution, together with the possibility of implementing the last two, makes mobility a more general concept and, hence, a good candidate for a primitive in this model of computation.

Another pair of concepts is individuality and what is from the collective, which is also relevant for global computers. Physically, every person is individually unique. There are refinements of physical characteristics that depend on genes, e.g. groups due to family factors. However, at the collective level, all people have common features, such as two eyes, two ears and one mouth. Accordingly, every individual has his or her own personality, and yet they share many collective standards: for instance, it is commonly felt that Marilyn Monroe was beautiful. These psychological standards vary according to place and time, and some of them change more slowly, others more quickly. Culture and fashion are two examples of collective standards. Family psychological characteristics, including those due to education, is an example of psychological characteristics

shared by groups. Rules of good behavior exemplify group or collective common sense, and is normally conscious behavior.

So far computation is relatively simple for us, computer scientists, but when one investigates deeply human psyche and starts thinking about unconscious, the subject becomes with synthetic nature. The psychologist Carl G. Jung[45], for instance, studied deeply what he called the *collective unconscious*, which is a theory based on science, but also a theory that contains elements from his philosophical view.

An analogy can be made between the hierarchy of characteristics as described above, and mobile agents systems. Because agents cannot view the internal parts of the interpreter implementation, and because all interpreters of the system should be the same or at least compatible with each other from a minimum extent, the interpreter corresponds to the collective unconscious, while the mobile agent corresponds to the conscious part of an individual. It turns out that such a psychological model can be somehow simulated by computers. The power of the collective unconscious can be illustrated in the following way: if there is some change or mistake in the implementation of the interpreter, like a social revolution, all mobile agents of the system may be critically affected at once. On the other hand, common sense and other kinds of information ought to be ubiquitous resources, provided by the mobile-agent system, i.e. mobile agents do not need to carry such established knowledge about the world, nor even any established belief system. Although the work of Jung form one of the most general psychological models, the Jung's model seems to apply to mobile agents, including the four psychological types, namely, reasoning, five-sense perceptions, intuition and feeling. The exceptions are probably intuition and feeling, although one can develop software which imitates human behavior.

Within the scope of AI, neural networks also make use of mobility. A neural network tends to represent what is learned from perception and possibly intuition. Using some mobile agent technology, a neural network may be spread out even over the globe, in such a way that the perception is also spread out, which contrasts with human perception. Because of this observation, this implementation can be seen a novel hybrid model of computing. One can observe that, whereas deductive systems are closer to the western way of thinking[47], neural networks with fuzzy systems[49] are opposite models closer to the eastern view, although this difference tends to disappear. I would like to stress that such applications use the technology of mobile agents to implement mobility, but *mobility is an essential property of networks in general*. And since the whole universe is on the move, mobility is a relative concept.

Regarding computation, as well as *mobile computation*, there is another modality of mobility, namely *mobile computing*[62]. The latter modality comes from wireless networks and portable computers, a subject somewhat close to robotics in a sense. Both forms are described in [18] or [19], and both are orthogonal to each other, for instance, one application does not affect the other[72], at least directly. As an example, to be general I must consider that a mobile agent can move from one craft to another, both on the move, and such a double movement is part of the general notion of computation.

5 An intuitive notion of computation

Traditionally, the models of computation are: Turing machines, Church's λ-calculus, Post production systems, Kleene's μ-recursion schemes, Herbrand-Gödel equational definability, Shepherdson-Sturgis register machines, the while programming language, and flow charts. Mobility is becoming part of candidates to the next generation of established models of computation.

Taking the example of on-line CPU loans or rentals, agents come to the host and, after identification and/or negotiation, use the CPU and possibly other resources, depending on the agreement. If it becomes expensive, some agents move to another host. Besides practical concerns, any application that depends on the presence of the "computational entity" acting locally is an example that there are computations which cannot be done without some form of interpreted code mobility. Should space and time are regarded, it is not difficult to find other examples. In particular, after having transported an agent to some virtual machine, while that agent interacts locally, some connections may be interrupted but the agent's computation might not be affected by that (temporary) interruption.

Therefore, in this paper, I define a notion of computation from an operational standpoint, in particular, I am concerned with time and space as part of a somewhat general notion and, therefore, part of the model that I shall present.

Thus, some conceptual issues are: *what is a computational entity? what do we mean by 'code' and 'interpretation'?* In some sense, 'code' can be any data, and the present discussion on computation leads one again to philosophy. There is the same for the term "executable code" which, depending on the adopted meaning, there are two different views in computer science. Therefore, code mobility introduces a philosophical view to computer science.

In [15], Cardelli briefly and informally defines *global computation* and points out several related issues, such as how multiple global computers can interact effectively. As he says, the main characteristic of global computation is the geographical distribution. Although every planet has its globe, the term global computation refers to this planet. Although I keep the term "global computer" from his article, I prefer to use the term global computing instead of global computation.

In this way, I shift the notion commonly referred to as "computation" to be referred to as *mathematical computation* to accommodate mobility as a primitive of the notion which I refer to as *computation*, for mobility is the focus of attention in this paper. Alternatively, one may prefer to refer to the same notion as *physical computation* while keeps the traditional meaning of computation.

Parallel computation is another abstract and theoretical concept that is modeled in π-calculus[55]. In that article the author shows that names of channels can be passed from one process to another, for instance, and that was a major step in the foundations of mobility and computation. Technologically, parallel computing has been linked to super-computing and powerful machines, but code mobility can also implement parallel computation over a local-area network or wide-area network or both. Thus, *parallel computation is simply computation in parallel.*

In [33], the authors introduce the distributed Join-calculus, which is an extension of the Join-calculus[32] for mobile agents. Both are asynchronous variants of π-calculus with the same expressive power as the latter, but the former

provide better locality and better static scoping rules[33]. To represent mobile agents, the distributed Join-calculus introduces locations, and for unreliable environments, that calculus also provides a simple model of failure.

Ambient calculus[18, 19] also captures the notion of code mobility, and, because the calculus is also partially based on the π-calculus, it also describes parallel processes. The Ambient model is also inspired by Telescript but almost dual to it, according to the authors. Other important contributions have been made since then, such as [14]. However, like other calculi and unlike Distributed Join-calculus and the Seal calculus[75] which is another extension of π-calculus, it does not abstract other details associated to mobility, such as resources and uncertainty, e.g. due to unreliability of physical media. On the other hand, the Seal calculus and others are somewhat practical calculi, in the sense that they are dependent on current structures such as the Internet. Although the notion of process[51] migration[56] is not new, the term *mobile computation* was coined by Cardelli in [16].

An interesting feature of a model of computation with strong mobility is that they *transcend* term rewriting systems. In comparison to models based on π-calculus, the **flyto** primitive (i.e. an instruction needed in every mobile agents programming language) moves not only the remaining symbols but also its surrounding context. Here I present an example of a rewriting system-like rule, in Ambient calculus, for moving an agent composed of two parallel processes ($\mathbf{flyto}(B).P$ and Q) from the place A to the place B:

$$A[\ (\mathbf{flyto}(B).P)|Q\ ]\ \parallel\ B[\ ]\quad \xrightarrow{\tau}\quad A[\ ]\ \parallel\ B[\ P|Q\ ]$$

The above rule cannot be applied using, for instance, a context-free grammar[5, 40, 43]. Moreover, at a more practical level, if one considers that places have their local resources and that agents typically use them locally, the order in which agents move *does* matter. For instance, in an unreliable environment, one cannot think in terms of a more general sense of the Church-Rosser property[37].

Parallel computation is a more general and abstract notion in comparison with recursive functions, that is, the latter is like a thiner granule. The linear operator $\otimes$, for instance, captures the notion of parallel operands, but it is still purely mathematical. Because I am looking for generality, my notion of computation includes both parallel computation and mobility, in addition to the physical[67] nature of this form of computation. In comparison to Ambient calculus, for instance, I consider timeouts in the model.

Although the mobility community in computer science established global computing for structures such as the Internet, certain issues related to computation are not limited to this planet. For example, a mobile agent can migrate from a spacecraft and continue its computation at another spacecraft no matter where they are. Agents can also travel from one planet to another no matter the distance between them. Because of this, I can perceive another term to refer to another model that includes code mobility. I use the term *physical computation* to stress the physical nature of computation, and *computation in the real world* to stress the philosophical, physical and psychological aspects of computation together. Additionally, I also use the term *computing in the real world* to include programming languages, technologies and applications.

Larger distances and time intervals for mobility are two fundamental characteristics of code mobility in comparison to the computation local to a single

hardware. In [15], Cardelli describes the main characteristics of global computing and I summarize them below:

- Parallel or concurrent processes.
- Code mobility.
- Latency and bandwidth are directly addressed.
- The availability of resources are distributed geographically, which requires that programmers be aware of locality of resources. This in turn replaces a established law in distributed systems.
- Higher level of interaction between users and machines.
- Security and privacy are particularly critical.

Thus, physical computation is a more abstract and theoretical notion than global computing, because physical properties of time and space are not limited to this planet or Internets. Physical computation and global computing almost entail mobility. However, while mobile code systems necessarily produce physical computation, this can be done locally or remotely, but not necessarily on a global environment.

If one starts considering mobile computation as a specialization of computation, we can see a kind of ontological paradox[20] in the traditional notion of computation: *from the moment that one conceives the idea of moving computation, one can observe that an effective general notion of computation transcends pure mathematics and meets the physical world*. Furthermore, objects in the real world are far from being perfect as the mathematical objects which one idealizes. In other words, code mobility changes the notion of computation.

Considering that humans have unconscious, the task of simulating human behavior with computers becomes dramatically more difficult. Here I give one more example of this view. It can be perceived that the sensation of *pleasure* conceptually is a mechanism of which the nature makes use to preserve both the individual and their species. At the individual level, a delicious (or beautiful) meal is sometimes able to create the wish in some person to feed themselves, sometimes even without any need. At the collective level, sexual pleasure can make a person pregnant. According to Jung, the human unconscious is often projected on things and people that one deals in one's daily life, and that mechanism can work individually or collectively. Thus, *projection* is a view that the individuals have of their unconscious, both personal and collective, and the rôle of projection seems to be to make them aware that the source of the standard of their interactions with the world is in themselves, and possibly that the standard needs change.

Projection is a mechanism of which the psyche makes use to advise the individual concerning what they do not perceive in themselves. In some sense, although projections are not normally pathological, the rôle of projection is somewhat similar to physical pain, which advises the individual that he or she is sick and, therefore, should seek treatment. Some other physical symptoms seem to play this rôle but they act at the collective level especially when the disease is contagious.

The collective unconscious makes sense as a *psychological* mechanism that seems to protect the individual and the species. To date, in addition to the approaches in natural sciences, there are different philosophical positions concerning the nature of psyche, some more complex than others. A recent selection on the rôle of analogy in the context of cognitive science is in [36].

To establish what computation is, I find a philosophical question: does any computation exist that cannot be perceived by humans at all? The answer also depends on the philosophical position. Although I use the term "real", I also adopt some idealistic ideas, i.e. computation in the real world regards human as a central component, but this does not exclude beliefs in God, either internal or external, and this is another subject in philosophy. The previous question can be used to provide foundations for a theoretical computer science based on physics and philosophy, as well as on mathematics. The relationship between mathematics and computer science is certainly very strong, like the relationship between mathematics and physics[31].

Here, as an example of computation, I present the reduction steps in some computation using some version of λ-calculus with some syntactic sugar:

$$(\lambda f.f\ 10)(\lambda x.x+1) \overset{\lambda}{\rightsquigarrow} (\lambda x.x+1)10 \overset{\lambda}{\rightsquigarrow} 10+1 \overset{\lambda}{\rightsquigarrow} 11$$

An interesting introductory and long study on λ-calculus and models of untyped λ-calculus is in [8]. Although these steps of computation are described as purely abstract objects, at the moment that I read them the *actual* computation is carried out mentally, in a context at some time and at some place.

6 A little on computability

Alan Turing demonstrated that the problem of determining algorithmically whether a Turing machine M_1 halts is unsolvable[11, 44]. This is one of the known theorems in computer science, often labelled as *the unsolvability of the halting problem*. The recent course book, [25], contains the well known proof of this theorem.

On the one hand, this problem is unsolvable because it is generally assumed that the tape is infinite, otherwise solutions to the halting problem would be like in chess, which regards a match a draw when the same position appears three times or in which the same sequence of moves happens three times. Accordingly, regarding that the computation by M_1 can be reproduced literally, one solution to the halting problem could be as follows:

```
—— Let the algorithm In interpret Turing machines.
halts := 0;
p := pointer(firstinstruction);
states := [ ]; —— comment: states now has the empty list.
while halts = 0 do
  ⟨operator, operands⟩ := fetch(p);
  p := nextinstruction(p);
  if operator is halt then
    halts := 1;
    states := states : "halt"; —— concatenates
  else
```

```
      ⟨newstate, p⟩ := execute(operator, operands);
      if newstate in states then halts := −1;
      else
        states := states : newstate;
      endif
    endif
  endwhile
  if halts is 1 then
    write "The analyzed program halts.";
  else
    write "The analyzed program loops."
  endif
```

where : is the concatenation operator. Thus, if the tape was finite there could be a finite number of configurations, all of which could be checked by the algorithm In, although this could still be a complex problem[44]. If a configuration repeated once, In would decide that the computation of M_1 was infinite[63], otherwise both machines would halt on normal termination. However, this is also a hypothesis. In this case, the time for solving the problem, as well as the above solution, are exponential (non-polynomial, in **NP**). However, apart from complexity issues, my observation is that no real machines have infinite-sized memories. Therefore, from the perspective of computability theory, it turns out that the halting problem is decided in the real world, although its solution is infeasible in some cases.

7 A notion of computation

Although there are other notions of computation, using local symbol definition, here I consider that computation can be ϕ in the following signature:

$$\phi : \pi \times \rho, \quad \rho : \tau \times \mathcal{U} \times \psi$$

where π is the sequential concept of computation, and ρ is an extra philosophical component, which in turn is defined as a product of time τ, space $\mathcal{U}$ and some possible psychological component ψ. Here, place and time can be those traditional physical dimensions, while ψ is with respect to some observer. In this piece of work, I do not use the above signature, which was shown as illustration.

7.1 A view of time, a representation

In many articles on temporal logics[1, 2, 35] commonly applied to AI planning systems[3] and other fields, time is often represented by using real values where, as time goes by, *the present moment* normally increases. There might be branches along these lines to represent "futures". There are other approaches, such as in [64] that can also be useful for applications, including system specification, but also to express natural sub-languages by using particular cases of modality.

In this paper, I adopt a form of representing time by making use of a flow. Thus, let us define $\mathbb{T}$ as an infinite set of temporal moments and let the flow be linear in $\mathbb{R}$. I use relational operators over the real numbers to state temporal relations.

The longer the distance is, the more significant modern physics is. Thus the present model is only a simplification of my intuitive notion i.e. this notion depends upon a number of factors that do not appear in these semantic rules, such as those caused by gravity and bodies, as well as what can be discovered in physics.

The operators over time instants refer to the daily-life temporal concepts, e.g. $a <_t b$ refers to *"a happens before b"* and so on. Apart from such order operations, there is no interval relationship over indexes. Thus, if $t \in \mathbb{T}$ is used as time variable, $t_j >_t t_i$ always holds for $j > i$ but $t_{i+1} -_t t_i$ is not necessarily equal to $t_{j+1} -_t t_j$. If $\mathbb{T}$ is $\mathbb{R}$, therefore $<_t$ is $<$ and $-_t$ is $-$ without further formalization.

I do not use $<_s$ here. $-_s : \mathbb{S} \times \mathbb{S} \longrightarrow \mathbb{S}$ is some approximation of the Euclidean distance from the first operand to the second one

$$(x_i, y_i, z_i) -_s (x_j, y_j, z_j) \stackrel{\text{def}}{=} \text{approx}\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2}$$

by using some mathematical method, already known for example.

One may want to represent time in a different way.

Here I define computation by defining the set of operations of some abstract machine.

7.2 States of the Real World

In this subsection, both space and time[9] are defined as continuum[28]. Much current theoretical work in computer science, such as [12], makes use of some form of continuous time. I choose and present one philosophical view as an example, having in mind that it is not necessarily a proposal for all. One subjective view is needed for supporting explanations in next subsection. Therefore, since such a model supports the idea of computing in $\mathbb{R}$, it may also support the idea of computing in $\mathbb{Z}$ with possible minor adaptation. In parallel to this, as an example that philosophy is a basis for computer science, in this subsection, I am regarding a hypothetical situation where agents travel to some country, for instance, China and, therefore, are culturally exposed to their very ancient wisdom. However, in India for instance, although it is another country in Asia, agents would be exposed to some different cultural background. In general, computer science has been based on modern western culture, but mobile agents on a global environment is making one think about the meaning of computation from different perspectives. Furthermore, on the one hand, complexity theory states what computers can do. On the other hand, ethics[46] (which is a branch of philosophy) asserts what computers should and what they cannot morally do, i.e. there are two complementary approaches. Internethics may be viewed as a good term for this new area.

Following this, during computation, each *real state*, or here I simply refer to it as "state", is not only the context of the program, in its traditional sense. It is also behaviors, including actions, which are unique in time and space.

Behavior is the part of a state that can be perceived externally, while the internal state contains the perception of the external world. A state is a set that can contain behavior and internal state, which in turn can contain locations, the characteristics of the physical world at a specific time, e.g. the whole mankind and all computers, the whole world, although not everything is accessible by the agent in question. In this sense, there is a mismatch between real states and the machine set of behaviors. The former is uncountable, the latter is countable[31]. Human experience is continuous in both space and time[1]. I tend to perceive time and space as in the structure of the set of real numbers.

Without defending a particular view, an *analogy* can be drawn between this discussion and in the relationship between I Ching and Taoism. I Ching hexagrams are pictorially composed of six lines. Some lines might move from *yin* to *yang*, others might move from *yang* to *yin*. Indeed, the I Ching is based on the binary system. On the other hand, although Taoism is also based on this pair of concepts, they are pictorially shown inside the Tao circle in such a way that one is gradually (but without granularity) being moved to the other. This analogy suggests that the I Ching is only a simplification of the Taoist view of reality as much as machines, by nature, are simplifications of the real world.

As another analogy, the classical musical language as well as the capability of some instruments can also be seen as simplifications of music. For some instruments, there are only 12 notes, although the scale is cyclical and can be repeated with higher or lower pitches. However, between any two subsequent notes rests a continuum interval of frequencies.

Likewise, from a somewhat similar point of view, although digital computers are useful and much can be improved on them, human computation might not be a concept limited to the set of integers, or perhaps it is better to define the notion of computation more precisely. That is, it is important to make clear up to what extent one is talking about computation, whether e.g. feeling and intuition are really computation, or whether this pair of psychological concepts and others, say *synthetic* concepts, can only be a matter of analytical simulation. Moreover, $\mathbb{R}$ certainly suggests the potential for future discoveries, as the infiniteness of the tape in the Turing machine model represents this potential, which is infinite. As a concrete example, if I want to conceive a circumference in a Cartesian plan, it is sufficient to have its equation, i.e. $x^2 + y^2 = r^2$ where r denotes its radius, but, to calculate its coordinates, I must convince myself that between any two points there exist infinite points. Although the infiniteness of the tape in the Turing machine model suggests this potential, proposing a model in $\mathbb{R}^4$ may be more natural and easier to conceive from a different view of computation. As one example of other work on a continuous three-dimensional space, in [42], the authors introduce an approach to the solution of the pursuit problem in the Euclidean $\mathbb{E}^3$ space.

It is known that humans reason and do research in science and mathematics in many ways, while the mathematical world is pure, exact or precise and perhaps very clean, tidy and structured in comparison to the psychological world, with dreams and the unconscious, for instance. An example of this kind of issue is input-output in purely functional languages, solved by using monads[38, 76], but a problem that can also be seen from a physical standpoint instead. There

[1]The notion of time might also be even individual, e.g. for a 2-year-old child, one year corresponds to the experience during half of life, and, perhaps because of this, I may feel that time goes by quicker as I get older.

are other different views regarding the match between mathematical and real worlds, and this is also another piece of evidence that computation is a philosophical notion. This almost entails that philosophy is a theoretical basis of computation and computer science.

To compute in $\mathbb{R}$ is not a novel idea, and there are good references such as [73].

In this discussion, I can still see a finite computation in a discrete interval in time as a sequence of states $s_{t_0}s_{t_1}...s_{t_n}$, where $\{t_0, ..., t_n\}$ are chosen instants, and I simply add the notions of time and space for every state. To define computation I need to define an abstract machine. My machine is a virtual machine that supports mobile agents, i.e. a virtual machine for a subset of an imperative language, such as Pascal or C, in addition to the ability to move the code, data and state of the computation in accordance with the definitions here.

Firstly, I define a state as a tuple composed of an internal state (ι), the external state of behavior (β), a place $p \equiv \langle x, y, z\rangle$ and an instant (τ). Thus,

$$s \in S,\ s \stackrel{\text{def}}{=} \langle \iota, \beta, x, y, z, \tau\rangle$$

where S is the set of all possible states. In this paper, I define ι and β as *sets* of propositions. I define the empty state as $\mho \in S,\ \mho \stackrel{\text{def}}{=} \langle \emptyset, \emptyset, 0, 0, 0, 0\rangle$. Thus, for $s_1 \equiv \langle \iota_1, \beta_1, x_1, y_1, z_1, \tau_1\rangle \wedge s_2 \equiv \langle \iota_2, \beta_2, x_2, y_2, z_2, \tau_2\rangle \wedge \tau = \tau_1 = \tau_2$, and ζ being a condition (a predicate as a singleton), there are six properties, which are the following:

P1:

$$\zeta \in s_1 \leftrightarrow \zeta \in \iota_1$$

That is, if a condition is in a state, it means that it is part of its internal state. Similarly, two predicates for sets:

P2:

$$s_1 \stackrel{s}{\subset} s_2 \leftrightarrow (\iota_1 \subset \iota_2 \vee \beta_1 \subset \beta_2) \wedge p_1 =_s p_2 \wedge \tau_1 =_t \tau_2$$

Accordingly, $s_1 \stackrel{s}{\supset} s_2 \leftrightarrow s_2 \stackrel{s}{\subset} s_1$.

P3:

$$s_1 \stackrel{s}{=} s_2 \leftrightarrow \iota_1 = \iota_2 \wedge \beta_1 = \beta_2 \wedge p_1 =_s p_2 \wedge \tau_1 =_t \tau_2$$

Accordingly, $s_1 \stackrel{s}{\subseteq} s_2 \leftrightarrow s_1 \stackrel{s}{\subset} s_2 \vee s_1 \stackrel{s}{=} s_2$.

P4:

$$s_1 \stackrel{s}{\neq} s_2 \leftrightarrow \iota_1 \neq \iota_2 \vee \beta_1 \neq \beta_2 \vee p_1 \neq p_2 \vee \tau_1 \neq_t \tau_2$$

and two functions for the spatial sets, **P5:**

$$s_1 \stackrel{s}{\cup} s_2 \equiv \langle \iota_1, \beta_1, x, y, z, \tau\rangle \stackrel{s}{\cup} \langle \iota_2, \beta_2, x, y, z, \tau\rangle \equiv \langle \iota_1 \cup \iota_2, \beta_1 \cup \beta_2, x, y, z, \tau\rangle$$

P6:

$$s_1 \stackrel{s}{\cap} s_2 \equiv \langle \iota_1, \beta_1, x, y, z, \tau\rangle \stackrel{s}{\cap} \langle \iota_2, \beta_2, x, y, z, \tau\rangle \equiv \langle \iota_1 \cap \iota_2, \beta_1 \cap \beta_2, x, y, z, \tau\rangle$$

I also take the liberty to use ε to denote the idle state. There is only one occurrence of the idle state in any computation. Notice that ε and $\mho$ are not the same notion.

In this paper, because ι and β are closely related, I collapse ι and β and refer to them as *virtual state*. Thus, from now on a state is the tuple $\langle r_t, x, y, z, t\rangle$ where r_t is the corresponding virtual state at the time t.

As already mentioned, although I informally consider the possibility of continuous flow-like multi-dimension set of states with respect to space and time, here I define a simplified version of computation, i.e. computation in the real world is a sequence of states, $s_{t_0} s_{t_1} ... s_{t_n}$ that one selects according to a particular focus of attention.

Let S be the set of all machine states, S isomorphic to $\mathbb{N}$. I define a (non-reflexive except for one case, and) anti-symmetric relation $\xrightarrow{c}: S \times S \longrightarrow Bool$ to indicate the existence of two subsequent states associated to the computation. I define $\xrightarrow{c}$ in terms of its properties as follows:

- $s \xrightarrow{c} s_1 \wedge s \xrightarrow{c} s_2 \Rightarrow s_1 \overset{s}{=} s_2$, $s_1 \xrightarrow{c} s \ \wedge \ s_2 \xrightarrow{c} s \Rightarrow s_1 \overset{s}{=} s_2$.
- $s \xrightarrow{c} s \Rightarrow s \overset{s}{=} \varepsilon$.
- Its particular case $\varepsilon \xrightarrow{c} \varepsilon$ which always holds.

I define $\xrightarrow{+c}$ as the transitive relation as follows:

$$s_{t_0} \xrightarrow{+c} s_{t_n} \overset{\text{def}}{=} (s_{t_0} \overset{s}{\neq} \varepsilon \wedge s_{t_0} \xrightarrow{c} s_{t_n} \xrightarrow{+c} \varepsilon) \ \underline{\text{v}} \ (\exists s_{t_1} \in S : s_{t_0} \xrightarrow{c} s_{t_1} \xrightarrow{+c} s_{t_n} \xrightarrow{+c} \varepsilon)$$

where $\underline{\text{v}}$ is the *exclusive or*, and $\forall s_0, s_1, s_2 \in S : s_0 \xrightarrow{+c} s_1 \xrightarrow{+c} s_2$ is defined as $s_0 \xrightarrow{+c} s_1 \wedge s_1 \xrightarrow{+c} s_2$ and the same holds with $\xrightarrow{c}$. Among others, three important properties of $\xrightarrow{+c}$ are the following:

- $s \xrightarrow{+c} s \Rightarrow s \overset{s}{=} \varepsilon$.
- $\forall s, s \xrightarrow{+c} \varepsilon$ which is my practical view of computation.
- $s_i \xrightarrow{+c} s_j \wedge s_i \xrightarrow{+c} s_k \Rightarrow (s_j \overset{s}{=} s_k \ \underline{\text{v}} \ s_j \xrightarrow{+c} s_k \ \underline{\text{v}} \ s_k \xrightarrow{+c} s_i)$, i.e. $\xrightarrow{+c}$ is unique with respect to $\xrightarrow{c}$.
- $s_i \xrightarrow{+c} s_j \wedge s_i \overset{s}{\neq} \varepsilon \Rightarrow \neg(s_j \xrightarrow{+c} s_i)$

In this way, *computation in the real world* can be defined as any finite sequence of states over the time, where every two subsequent states are linked with an application of $\xrightarrow{c}$ relation, where the last state of that sequence is the only inactive state of that computation. The intention might be the same, the place might be the same but, if the instants are not the same, the external world is no longer the same. Therefore, from this point of view, computations performed at different times cannot be the same. With some simplification, human thought can be an example of $\xrightarrow{+c}$, where ε corresponds to the individual's death. It is very difficult, if possible, if I want to mathematically establish when a computation starts and when it finishes for this case. So I state in this example that human computing starts when they are born and finishes when they die, and it is also always finite. I can also think in terms of collective computing which may never finish due to (human) communication, or may finish due to some colliding asteroid, for instance.

An analogy can be made between computation and a melody being played, where not only the sequence of notes is relevant but also the duration of every note among other variables. More generally, there is a subtle difference between

a melody and its performance, whereas, likewise, there is some subtle difference between the mathematical and physical forms of computations.

In comparison with any rewriting system[6], except for the idle-state case here, although $\xrightarrow{+c}$ is also transitive, such a relation is neither symmetric nor reflexive as time never goes by backwards nor it stops, i.e. joining together two of the above properties we obtain the following one:

$$s_{t_i} \xrightarrow{+c} s_{t_j} \Rightarrow (s_{t_i} \overset{s}{=} s_{t_j} \overset{s}{=} \varepsilon) \ \underline{\mathrm{v}} \ (s_{t_i} \overset{s}{\neq} s_{t_j} \wedge \neg(s_{t_j} \xrightarrow{+c} s_{t_i}))$$

Moreover, this relation implies some measure of uncertainty due to the somewhat unpredictable nature of the real world. While any computation happens, the probability of its success gradually increases over time. To add some probability between states is enough to introduce a more general and physical model, in some sense. Hence, I alternatively define $\xrightarrow{+c}$ as follows:

$$\begin{aligned} s_{t_0} \xrightarrow{+c} s_{t_n} \overset{\text{def}}{=} \exists m \in \mathbb{R}, 0 \leq m \leq 1 : (s_{t_0} \overset{s}{\neq} \varepsilon \wedge \Psi(m : s_{t_0} \xrightarrow{c} s_{t_n}) \xrightarrow{+c} \varepsilon) \ \underline{\mathrm{v}} \\ (\exists s_{t_1} \in S : \Psi(m : s_{t_0} \xrightarrow{c} s_{t_1}) \xrightarrow{+c} s_{t_n} \xrightarrow{+c} \varepsilon) \end{aligned} \tag{1}$$

where $\Psi(n : \varphi)$ represents the assertion φ with probability n. Therefore, $\forall s : \Psi(1 : s \xrightarrow{+c} \varepsilon)$ and also

$$\begin{aligned} &\forall s_0, s_1, s_2 \in S, mn \in \mathbb{R}, 0 \leq mn \leq 1 : \Psi(mn : s_0 \xrightarrow{+c} s_2) \overset{\text{def}}{=} \\ &\exists m, n : \mathbb{R}, 0 \leq m \leq 1, 0 \leq n \leq 1, mn = m \times n : \\ &\Psi(m : s_0 \xrightarrow{+c} s_1) \wedge \Psi(n : s_1 \xrightarrow{c} s_2) \end{aligned} \tag{2}$$

The $\Downarrow\!\Downarrow$ relation indicates that two states coexist independently. Using the present style...

$$\begin{aligned} s_1 \Downarrow\!\Downarrow s_2 \overset{\text{def}}{=} & s_1 \equiv \langle r_1, x_1, y_1, z_1, \tau_1 \rangle \wedge \\ & s_2 \equiv \langle r_2, x_2, y_2, z_2, \tau_2 \rangle \wedge \\ & (s_1 \overset{s}{\cap} s_2 \overset{s}{=} \mho \wedge (x_1 \neq x_2 \vee y_1 \neq y_2 \vee z_1 \neq z_2)) \end{aligned} \tag{3}$$

Let $\mathbb{U}$ be the set of computations and S be the set of states. I define the function $\overset{s}{\rightsquigarrow}$: $\mathbb{U} \times S \longrightarrow \mathit{Bool}$, which informs whether the second operand is the last non-idle state *during* a computation, which in turn is given as the first operand.

The function $\overset{s}{\rightsquigarrow}$ may have included the influence from the behavior of other computations, not only interaction with other computations. Thus, a program running twice produces two different computations (and perhaps two different behaviors). There are two more significant properties:

$$(\forall C_1, C_2 \in \mathbb{U}, \forall s \in S) \ C_1 \overset{s}{\rightsquigarrow} s \wedge C_2 \overset{s}{\rightsquigarrow} s \Rightarrow C_1 \overset{c}{=} C_2$$

where $C_1 \overset{c}{=} C_2 \overset{\text{def}}{=} \exists n \in \mathbb{N} : C_1 \equiv s_0 s_1 \ldots s_n \wedge C_2 \equiv s'_0 s'_2 \ldots s'_n \wedge \forall (i \in \mathbb{N}, i \leq n) : s_i \overset{s}{=} s'_i$. And also:

$$(\forall C \in \mathbb{U}, \forall s_1, s_2 \in S) \ C \overset{s}{\rightsquigarrow} s_1 \wedge C \overset{s}{\rightsquigarrow} s_2 \Rightarrow s_1 \overset{s}{=} s_2$$

That is, computation has the property of being a unique object with respect to its final state. I also define α as the set of all programs, i.e. the set of all mobile agents (or mobile processes), which are the potential for computation. Let $\mathbb{U}$ be the set of computations and S be the set of states. The function $\lceil\ \rceil : \mathbb{U} \longrightarrow S$, that indicates the last active state, is defined as the following:

$$\lceil C \rceil = \begin{cases} s_0 & \text{if } C \equiv s_0\varepsilon, \text{ where } s_0 \neq \varepsilon \wedge s_0 \xrightarrow{c} \varepsilon \\ s_n & \text{if } C \equiv s_0 ... s_n\varepsilon, \text{ where } ((\forall i \in \mathbb{N})\ s_i \neq \varepsilon) \wedge s_n \xrightarrow{c} \varepsilon \end{cases}$$

for every computation C which entails some $n \in \mathbb{N}$.

7.3 The Brief Definition of the Space-Time Logic, @-Logic

This section introduces the concise definition of the space-time classical logic, called here the @-logic.

For this paper, let $\mathbb{C}$ be the set of all formulae in the space-time logic, i.e. the language of the present logic. The syntax can be defined as follows:

Definition 1 *Let φ and ϕ be two formulae and α be a variable (a quantifier). Thus, the grammar for the space-time classical logic can be as follows:*

$$\varphi \longmapsto P \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \Rightarrow \varphi \mid \varphi \Leftrightarrow \varphi \mid (\exists\alpha)\ \varphi \mid (\forall\alpha)\ \varphi$$
$$\varphi \longmapsto @s \cdot t[\varphi] \mid \phi \mid @s \cdot t[\ ]$$
$$\phi \longmapsto @s \cdot t[\![\varphi]\!]$$
$$\alpha \longmapsto x \mid y \mid ...$$

where φ is the starting symbol, P stands for a proposition or predicate, α denotes a quantified variable in the @-logic, $s \in \mathbb{R}^3$ or $s \subseteq \mathbb{R}^3$ and $t \in \mathbb{R}$ or $t \subseteq \mathbb{R}$, depending on the focus of attention, points or sets. Let ϕ stand for semantics, which can be seen as a function whose domain is in $\mathbb{C}$ and semantic image is a parameter of the present logic. Both the syntax and semantics are simple: if φ is a formula, then

$$@s \cdot t[\varphi]$$

is a formula in the present logic, where s indicates the place where φ holds, and t indicates the time when φ holds. A particular case is

$$@s \cdot t[\ ]$$

which intuitively indicates that there is no assertion for space s and time t. This notation is capable of representing an empty data base or theory, for instance.

With $@s \cdot t[\![\varphi]\!]$, we are able to express "the meaning of φ at place s and time t".

Note that this language implicitly introduces a conjunction between the space and the time, for every space-time formula. Now, for any formula or expression, everything happens intuitively in the same way as it would happen in the classical logics, except that now there are the variables of space and time, and that the classical logic formula or expression is valid in the new context.

7.4 The Present Semantics of Computation

In this subsection, I attempt to formalize a simplified version of the previous notion of computation, informally introduced above in this section, by using the @-logic.

Let π be a program in some object language with some computation ϕ_c and let all of my definitions in this subsection apply to the scope π, unless stated otherwise. To avoid being exhaustive, I consider that variables have their separate scopes in each rule although they have the same names and meanings, except for variables defined explicitly as global for all rules. Let $\mathcal{A}^R$ be isomorphic to $\mathbb{R} \cup \{\boldsymbol{uu}\}$ and $\mathcal{A}^L$ stand for the set of Boolean values, and use these sets as carriers of the algebra[29, 54] that is going to be defined here.

I also write $\langle x, y, z\rangle$ explicitly when I want to stress the relationships between each of these coordinates and the time, although I do not use them individually in the semantic rules. There is one global definition, in symbols, $p \stackrel{\text{def}}{=} \langle x, y, z\rangle$ and it may be indexed, e.g. $p_i \stackrel{\text{def}}{=} \langle x_i, y_i, z_i\rangle$. As well as the notation in rules, I make implicit use of first-order predicate logic in these rules. When t is not the present the predicate expression does not hold. Thus, the states of the predicate expressions change as time passes. Thus, in the @-logic, I write logical expressions as well as operational expressions.

The present model is over the following definitions:

- There is a common three-dimensional space in $\mathbb{E}^3$, which is the universe. I equate $\mathbb{R} \equiv \mathbb{E}$ and use Cartesian coordinates $\langle x, y, x\rangle$ to refer to the points in (Euclidean) space $\mathbb{E}^3$. Thus, let $\mathcal{U} \stackrel{\text{def}}{=} \mathbb{E}^3$, be the universe;
- There is an infinite but countable set of possible agents $\triangle \stackrel{\text{def}}{=} \{a_1, ..., a_i, ...\}$ written in the language α;
- Every agent has its input queue;
- Every point $\langle x, y, z\rangle$ in $\mathcal{U}$ contains a finite set $Obj \stackrel{\text{def}}{=} \langle v, \{a_1, ..., a_n\}\rangle$ where $v \in Val, Val \equiv \mathcal{A}^R$, that is, v stands for either a real number or the $\boldsymbol{uu}$ value (which stands for *unknown* or a kind of vac*uu*m), and $\{a_1, ..., a_n\} = Y \subseteq \triangle$.

Then, I define the following Σ-algebra

$$\begin{gathered}\mathcal{A} \stackrel{\text{def}}{=} \langle \mathcal{A}^R, \mathcal{A}^L, Var, SObj, Loc, S, \mathcal{U}, 0, 2, \boldsymbol{uu}, par, p, p_0, p_i, q,\\ r, r_0, r^p, r^{p_0}, r_t, r_{t_0}, r_{t_0+I_b}, t_1, r_{t_i}, s, s', t, t_0, t_\varepsilon, t_f, t_i, \Delta t,\\ A_t^p, A_{t_i}^p, A_{t_0}^{p_0}, A_t^{p_0}, A_t, A_{t_0}, A_{t_i}^{p_i}, A_{t_0+I_b}^{p_0}, A^d,\\ u, v, V, x, y, z, x_0, y_0, z_0, x_i, y_i, z_i,\\ \Delta l, \Delta l_1, \Delta l_2, \Delta s, \Delta s_1, \Delta s_2,\\ I, I\mu, I_a, I_{a_1}, I_{a_2}, I_b, I_e, I_i, I_{:=}, IC, I_{th}, I_s, I_{wem}, \omega, \Psi\rangle\end{gathered}$$

for signature Σ here, where Var is the set of all variables internal to ϕ_c, and also to π; and since $Obj = \langle v, Y\rangle$ where $v \in \mathcal{A}^R$, $Y \subseteq \triangle$ as stated, $SObj = \mathcal{A}^R \times \mathcal{P}(\triangle)$; Loc is the set of internal locations, and S is the set of states. Let $u, v \in Val,\ V \in Var,\ p \in \mathcal{U}, r, s \in S, t \in \mathbb{R}$. Then,

$$\Sigma \stackrel{\text{def}}{=} \langle \{\mathcal{A}^R, \mathcal{A}^L, Var, Val, Loc, S, SObj\}, F\rangle$$

where F is consisted by $+$, $-$, $/$, $=$, $\neq$, $<$, $>$, $\wedge$, $\vee$, r^2 as usually defined in mathematics plus the following functions:

$$(locate) \quad \gamma : Var \longrightarrow Loc$$

$$(lookup) \quad \rho : S \times Loc \longrightarrow Val$$

$$(update) \quad \Delta \colon S \times Loc \times Val \longrightarrow S$$

$$(value) \quad \Theta : S \longrightarrow SObj$$

$$(first) \quad \xi : S \longrightarrow \mathcal{A}^R \times S$$

$$(del\ queue) \quad \chi : S \longrightarrow S$$

$$(ins\ queue) \quad \eta : S \times \mathcal{A}^R \longrightarrow S$$

$$(fault) \quad f\pi : \mathbb{R}^3 \times \mathbb{R} \longrightarrow \mathcal{A}^L$$

$$(physical\ move) \quad \ddot{\mu}x, \ddot{\mu}y, \ddot{\mu}z : \ \mathbb{R} \longrightarrow \mathbb{R}$$

Intuitively, γ maps a variable to its location; ρ results in the content of a location in some particular state; Δ updates the memory according to its parameters: location and value; Θ, provided a state, results in the value of the corresponding point in space and time; ξ gives the first element of the input queue and the state after the operation; χ removes the first element from this queue; η inserts a value as the last element of this queue; $f\pi$ is a predicate that, given some point p and time t, informs whether there is some fault (e.g. the presence of another agent) between the current point (an implicit parameter that denotes the point at the three current coordinates of the running agent) and p at t; and $\ddot{\mu}x, \ddot{\mu}y, \ddot{\mu}z$ are postfix functions that, given one coordinate results in the new corresponding coordinate due to possible physical hardware movement. That is

$$p\ddot{\mu} \stackrel{\text{def}}{=} \sqrt{(x\ddot{\mu}x)^2 + (y\ddot{\mu}y)^2 + (z\ddot{\mu}z)^2} \quad \text{(for an approximation of the real)}$$

For each semantic rule, as it is already clear here, to simplify a little the notation, I shall assume that the existence of more than one occurrences of the functions $\ddot{\mu}x, \ddot{\mu}y, \ddot{\mu}z$ and $\ddot{\mu}$ is not relevant, that is, I do not formally consider that the physical shifts of the machine for this tiny intervals might be at different velocities. However, in this case, to state this independence of velocities, I could index the occurrences of theses operators, for example, $\ddot{\mu}x_1, \ddot{\mu}x_2, \ddot{\mu}_1, \ddot{\mu}y_1, \ddot{\mu}_2, \ddot{\mu}y_2$ and so on. Additionally, let $s, s', r, r_t, r_0, r_{t_0}, r_{t_0+I_b}, r_{t_i}, r^p, r^{p_0} \in S$, as well as let r_0 be the initial state of the computing agent.

For the level of abstraction to capture computation, I regard the meaning of defining operations in terms of some sequence in some microcode. There are other alternatives. My choice is to divide the rules into two levels. One of them concerns the object of computation, as usual, and I call *object rules*. The other concerns the application of the object rules. This applies only for interaction and mobility operations. Let us start with the three rules as follows:

$$\ddot{a}_{t_0} \wedge (\forall t, \ (t_0 <_t t <_t t_0 +_t q_i) \Rightarrow \ddot{a}_t) \wedge K_{t_0+q_i} \wedge \forall t, t >_t t_0 +_t q_i, \ (\neg\ddot{a}_t \wedge \neg K_t)$$

where $\ddot{a}_t$ is the attempt to perform a remote operation (**wemove** although for this level of detail there is no timeout, **flyto** or **view** or **throw**) at time t by

using the rule 6 (**wemove**) or 7 (**flyto**) or 10 or 13 (both rules for **view**), or 16 (**throw**); K_t is the action of skipping the executing operation of rule 9 (**flyto**) or 15 (**view**) or 18 (**throw**) at time t; $q_i = \min(q, d)$, d is the delay due to the operation in question, q is the timeout for this occurrence of operation.

I also need to formalize the notion "the sooner the better". Thus, if t_i and t_j are time variables, $\ddot{a}_t$ is some action of type A to be performed at t, then the rule

$$(\ (t_i \leq_t t_j) \wedge \Diamond \ddot{a}_{t_i} \wedge \Diamond \ddot{a}_{t_j}\ ) \rightarrow \Box \ddot{a}_{t_i}$$

states that, given the above conditions, the action $\ddot{a}_{t_i}$ must be performed first.

Every state is a tuple $\langle r_t, x, y, z, t\rangle$ where r_t is the virtual state at some time t, and $\langle x, y, z\rangle$ are Cartesian coordinates at that locality. I use the notation $r[V/u]$ to mean that the variable V contains the value u in the virtual state r. I adopt A_t^p, for *ambient state*, to refer to a particular state at some place p and time t. Ambient states are important to stress the mobility of virtual states. Therefore, $r_t \cap A_t^p$ simply indicates that the virtual state r_t is placed in the ambient A_t^p at place p and time t, once the premise $r_t \subseteq A_t^p$ exists. I use this notation for mobility and other related operations, and simply write r_t to denote the same situation in other rules. In the semantic rules, I use the definition $s \stackrel{\text{def}}{=} \langle r, x, y, z, t_0\rangle$ which can also be represented as $@p \cdot t_0[r]$ in the @-logic. Thus, two or more states $s, s'...$ can happen at the same time, i.e. in parallel. In this case, I denote this as $s \Downarrow s' \Downarrow ...$.

Given that $\varepsilon \xrightarrow{+c} \varepsilon$, any agent segment of computation can also be composed in $S_1 \| S_2$ i.e. in parallel:

$$\frac{S_1 \models \langle r_t^{p_1}, x_1, y_1, z_1, t\rangle \xrightarrow{+c} s_{t_i} \quad S_2 \models \langle r_t^{p_2}, x_2, y_2, z_2, t\rangle \xrightarrow{+c} s'_{t_i}}{S_1 \| S_2 \models (\langle r_t^{p_1}, x_1, y_1, z_1, t\rangle \Downarrow \langle r_t^{p_2}, x_2, y_2, z_2, t\rangle) \xrightarrow{+c} (s_{t_i} \Downarrow s'_{t_i})}$$

where S_1 and S_2 denote two segments of computation for some t_i. For any two computations, the rule for composing them is different from the previous one:

$$\frac{C_1 \models \langle r_1, x_1, y_1, z_1, t_1\rangle \xrightarrow{+c} s_{t_i} \quad C_2 \models \langle r_2, x_2, y_2, z_2, t_2\rangle \xrightarrow{+c} s_{t_j}}{C_1 | C_2 \models (\langle r_1, x_1, y_1, z_1, t_1\rangle \Downarrow \langle r_2, x_2, y_2, z_2, t_2\rangle) \xrightarrow{+c} (s_{t_i} \Downarrow s_{t_j})}$$

where C_1 denotes a computation that finishes at t_i and C_2 denotes a computations that finishes at t_j.

Parallel computation presents behaviors in parallel. Notice that computations might interfere with each other.

My abstract model of computation is an extension of the while language, which is an abstract model of computation. Nonetheless, although that model has the *if* statement, I observe that this statement can be definable: that is, for p and q as a Boolean expression and statement, respectively, IF p THEN q is defined as

$x := 0;$
while $x = 0 \wedge p$ **do**
  $q;$
  $x := 1;$
endwhile

for a new variable x.

For defining the *if-then-else*, roughly speaking the reader can view two similar loops, and this change by removing *if* from the present model will simplify my effort. I am formulating a model in terms of one set of operations. It is important to see that these operations do not necessarily correspond to their suitability in programming languages that permit proactive move because here I am only defining an abstract model.

The coordinates $\langle x, y, z\rangle$ of $\mathbb{E}^3$, for instance, are not normally provided at the language level, but they are in the model because they provide a suitable notion of space for my purpose.

In this way, the operations form a somewhat minimal set of primitives for the present semantics of computation, i.e. it corresponds to an extension of the while language presented in the literature as in [39, 48, 57, 59, 70, 78, 79]). Thus, I formalize the operational semantics that complement theirs and give a few local primitives. Thus my working objects are:

- Real constants.
- The space initialization.
- The **create** statement, which creates another copy of the computation at another place.
- The arithmetical operators: $+, - : \mathbb{R} \times \mathbb{R} \longrightarrow \mathbb{R}$.
- The relational operators: $=, <: \mathbb{R} \times \mathbb{R} \longrightarrow \mathbb{R}$.
- The assignment statement.
- The **while** statement.
- Intentional unity mobility: the **wemove** statement.
- Broadcasting mobility: the **SpreadOut** statement.
- Strong mobility: the **flyto** statement.
- Communication: the **view** and **throw** statements for remote communication, and **read** and **write** for local communication.
- The *halt* operation. The $f\pi$ predicate.
- Physical mobility: no operators, but the $\ddot{\mu}x$, $\ddot{\mu}y$ and $\ddot{\mu}z$ postfixed functions defined in the above algebra. Further,

$$p\ddot{\mu} \stackrel{\text{def}}{=} \sqrt{(x\ddot{\mu}x)^2 + (y\ddot{\mu}y)^2 + (z\ddot{\mu}z)^2},$$

  more precisely, some approximation of this number.

Variables can range over an address space, that is, $Var \stackrel{\text{def}}{=} \{V_0, V_1, ...\}$, but here I shall use only the V symbol to denote any variable in Var.

From the **while** statement, one builds the *if-then*, *if-then-else* and *case* as I did or in the same manner. From these definitions, one builds *min* and *max*, then builds the logical operators, and so on.

Now I can present the initial axiom before one places agents and values in $\mathcal{U}$:

$$Initial: \quad \frac{}{(\exists t_0 \in \mathbb{T})\ (\forall s \in S,\ \forall t \in \mathbb{T},\ t_0 \leq_t t)\ s \equiv \langle r_{t_0}, x, y, z, t_0\rangle,\ \Theta s = \langle \boldsymbol{uu}, \emptyset\rangle}$$

and another rule with the $\overset{\text{eval}}{\rightsquigarrow}$ relation:

$$\frac{\rho(s, \gamma V) = u}{@p \cdot t_0[\![V, s]\!] \overset{\text{eval}}{\rightsquigarrow} @p\ddot{\mu} \cdot t_0 +_t \Delta t[u]}$$

where Δt is the time for accessing a variable at place $\langle x, y, z\rangle$ and time t_0. Notice that, in the signature, I defined $\gamma : Var \longrightarrow Loc$ as the function that, given a variable, locates its storage.

An agent can create another agent by executing the **create** statement. The new agent is a clone but it executes from scratch at a given coordinate. The semantics for this statement can be as follows:

$$\frac{r^{p0}_{t_0} \not\subseteq A^p_{t_i} \quad r^{p0}_{t_0} \subseteq A^p_{t_i+\Delta t} \quad @p_0 \cdot t[t <_t t_0 +_t q \wedge \neg f\pi(p,t)]}{\begin{array}{c} @p_0 \cdot t_0[\![\textbf{create}\langle x, y, z\rangle\ \textbf{time}\ q, r^{p_0}_{t_0} \cap A^{p_0}_{t_0}]\!] \overset{\text{exec}}{\rightsquigarrow} \\ @p_0\ddot{\mu} \cdot t[r^{p_0}_{t_0} \cap A^{p_0}_{t_0}] \downdownarrows @p \cdot t_i +_t \Delta t[r^p_0 \cap A^p_t] \end{array}} \tag{4}$$

where $t_i =_t t_0 +_t \Psi +_t I_c$, I_c is the time for interpreting the **create** statement. Ψ is defined globally as

$$\Psi = \text{approx}\ \frac{\sqrt{(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2}}{\omega}$$

where ω is the velocity of light and, for the rule 4, $t \geq_t t_i +_t \Psi$ and Δt is the time spent installing the agent at the destination, once the transmission has completed.

For the purpose of simplification and visibility, I allow myself a slight abuse of notation by not writing the temporal subscript in operations, such as $+_t$, when they are already part of a temporal expression appearing in subscript of a formula, for instance $A^p_{t_i+\Delta t}$, which appears above. The same is valid for spatial relations in subscript, as the meanings of the operators are clear.

The semantics for the corresponding timeout situation is as follows:

$$\frac{@p_0 \cdot t[t =_t t_0 +_t q \wedge f\pi(p,t)]}{@p_0 \cdot t_0[\![\textbf{create}\ p\ \textbf{time}\ q, r^{p_0}_{t_0} \cap A^{p_0}_{t_0}]\!] \overset{\text{exec}}{\rightsquigarrow} @p_0\ddot{\mu} \cdot t_0 +_t I_c[r^{p_0}_{t_0} \cap A^{p_0}_{t_0}]} \tag{5}$$

The *halt* operation is not a statement of the present language, that is, after the execution of the last statement in the program, the computation goes to the idle state. Although it is not a statement, I regard as if it were, as, here, I am interested in the computation itself:

$$(\exists k \in \mathbb{N})\ \frac{}{@p_0 \cdot t_0[\![halt, r]\!] \overset{\text{exec}}{\rightsquigarrow} @p_0\ddot{\mu} \cdot t_0 +_t k[\varepsilon]}$$

where *halt* corresponds to the implicit last operation.

For unity (CE or hardware) mobility, the command **wemove** moves the set of agents from the current point $p_0 : \mathbb{S}$ to another $p : \mathbb{S}$ (coordinates $\langle x, y, z\rangle$) as a unity, along with the corresponding value, given the coordinates of the destination. I assume that sets of agents can share a common place and move slower or at the speed of light, but robots movements can be a particular case of this form of mobility as long as I formalize some physical laws as well as constrain and regard the destination close enough to the source place for uniform and straight movements, in such a way that more complex movements could be obtained from a sequence of this simpler physical movements here. Thus, let p_0 represent the coordinates of the original place. While the unity moves (say, after some delay ζ), the original place becomes without value in the problem domain. The subexpression $\frac{2\times(p-_s p_0)}{\omega}$, below, represents the minimum interval with value of twice the space through which the light would traverse: Time for observation and, then, for moving the unity; I_{wem}, for interpreting the **wemove** command:

$$\frac{\begin{array}{c}\exists(t \geq_t t_0 +_t I_{wem} +_t \frac{2\times(p-_s p_0)}{\omega}) \\ p \equiv \langle x, y, z\rangle \wedge \Theta\langle r, x, y, z, t\rangle = \langle u, A^d\rangle \wedge \neg f\pi(p, t)\end{array}}{@p_0 \cdot t_0[\![\textbf{wemove } p, A^{p_0}_{t_0}]\!] \overset{\text{exec}}{\rightsquigarrow} @p_0 \cdot t_0 +_t \zeta[\langle \boldsymbol{uu}, \emptyset\rangle] \downdownarrows @p \cdot t[A^d \cup A^{p_0}_{t_0}]} \tag{6}$$

where $p_0 \equiv (x_0, y_0, z_0)$. For strong mobility, given the source and destination ambient states A^{p_0} and A^p, time t_0 and the current state $\langle r, x_0, y_0, z_0, t_0\rangle$, an operational semantics for the **flyto** p **time** q statement, which moves the computation to the position $\langle x, y, z\rangle$ using a set timeout q, can be as follows:

$$\frac{@p_0 \cdot t_0[r \subseteq A^{p_0}_{t_0}] \quad @p \cdot t[r \subseteq A^p_t] \quad @p_0 \cdot t[t -_t t_0 <_t q \wedge \neg f\pi(p, t)]}{@p_0 \cdot t_0[\![\textbf{flyto } p \textbf{ time } q, r \cap A^{p_0}_{t_0}]\!] \overset{\text{exec}}{\rightsquigarrow} @p \cdot t[r \cap A^p_t]} \tag{7}$$

where $(p_0 \neq_s p \wedge r \not\subseteq A^{p_0}_t)$ $\underline{\vee}$ $(p_0 =_s p \wedge r \subseteq A^{p_0}_t)$ and

$$t \geq_t t_0 +_t \Psi +_t \Delta l +_t \Delta s +_t I\mu \tag{8}$$

where Δl is latency, Δs is the time interval due to the code size, and $I\mu$ is the time for interpreting the **flyto** instruction. The semantics of the timeout for this operation is as follows:

$$(\exists!\ t \in \mathbb{T})\ \frac{@p_0 \cdot t_0[r \subseteq A^{p_0}_{t_0}] \quad @p_0 \cdot t_0 +_t q[r \subseteq A^{p_0}_t \wedge f\pi(p, t_0 +_t q)]}{@p_0 \cdot t_0[\![\textbf{flyto } p \textbf{ time } q, r \cap A^{p_0}_{t_0}]\!] \overset{\text{exec}}{\rightsquigarrow} @p_0 \cdot t_0 +_t q[r \cap A^{p_0}_t]} \tag{9}$$

Here is the semantics for the + operation:

$$\frac{\begin{array}{c}@p_0 \cdot t_0[\![a_1, s]\!] \overset{\text{eval}}{\rightsquigarrow} @p' \cdot t_0 +_t I_{a_1}[\langle u, s\rangle] \\ @p' \cdot t_0 +_t I_{a_1}[\![a_2, s]\!] \overset{\text{eval}}{\rightsquigarrow} @p \cdot t_0 +_t I_{a_1} +_t I_{a_2}[\langle v, s\rangle]\end{array}}{@p_0 \cdot t_0[\![a_1 + a_2, s]\!] \overset{\text{eval}}{\rightsquigarrow} @p\ddot{\mu} \cdot t_0 +_t I_{a_1} +_t I_{a_2} +_t I[\langle u + v, s\rangle]}$$

where s is any state, I is the time for evaluating the operation once the machine has the operands. $\boldsymbol{uu}$ permits that almost all operators of the underlying machine are lazy, according to the following rule:

$$\frac{@p_0 \cdot t_0[\![a_1, s]\!] \overset{\text{eval}}{\rightsquigarrow} @p \cdot t_0 +_t I_{a_1}[\langle \boldsymbol{uu}, s\rangle]}{@p_0 \cdot t_0[\![a_1 + a_2, s]\!] \overset{\text{eval}}{\rightsquigarrow} @p\ddot{\mu} \cdot t_0 +_t I_{a_1} +_t I[\langle \boldsymbol{uu}, s\rangle]}$$

The pair of rules for $-$, $=$ and $<$ are the same as for $+$, except that the operator is different. Notice that I am assuming that the above operands do not have any side-effect and, therefore, their evaluations do not move the computation, although the mobility can exist due to physical movement, $\ddot{\mu}$. Now here the assignment statement:

$$\frac{@p_0 \cdot t_0[\![a, s]\!] \overset{\text{eval}}{\rightsquigarrow} @p \cdot t_0 +_t I_e[\langle u, s\rangle] \quad @p \cdot t_0 +_t I_e[\Delta(s, \gamma V, u) = s']}{@p_0 \cdot t_0[\![V := a, s]\!] \overset{\text{exec}}{\rightsquigarrow} @p\ddot{\mu} \cdot t_0 +_t I_a +_t I_{:=}[s']}$$

where $s' \overset{\text{def}}{=} \langle r[V/u], x\ddot{\mu}x, y\ddot{\mu}y, z\ddot{\mu}z, t_0 +_t I_a +_t I_{:=}\rangle$, and I_a is the interpretation time for evaluating the expression, and $I_{:=}$ is the time for assigning the value u to the variable.

The other local expressions and statements are similar to the above. A semantics for the **while** statement is as follows:

$$\frac{\begin{array}{c} r_{t_0} \subseteq A^{p_0}_{t_0} \quad r_t \subseteq A^{p}_{t} \quad @p_0 \cdot t_0[\![b, r_{t_0} \cap A^{p_0}_{t_0}]\!] \overset{\text{eval}}{\rightsquigarrow} @p_0\ddot{\mu} \cdot t_0 +_t I_b[\langle v, r_{t_0+I_b}\rangle] \\ v \neq 0 \quad @p_0\ddot{\mu} \cdot t_0 +_t I_b[\![C, r_{t_0+I_b} \cap A^{p_0}_{t_0+I_b}]\!] \overset{\text{exec}}{\rightsquigarrow} @p_i \cdot t_t[r_{t_i} \cap A^{p_i}_{t_i}] \\ r_{t_i} \subseteq A^{p_i}_{t_i} \quad @p_i \cdot t_i[\![\textbf{while } b \textbf{ do } C, r_{t_i} \cap A^{p_i}_{t_i}]\!] \overset{\text{exec}}{\rightsquigarrow} @p \cdot t[r_t \cap A^{p}_{t}] \end{array}}{@p_0 \cdot t_0[\![\textbf{while } b \textbf{ do } C, r_{t_0} \cap A^{p_0}_{t_0}]\!] \overset{\text{exec}}{\rightsquigarrow} @p \cdot t[r_t \cap A^{p}_{t}]}$$

where $t_i =_t t_0 +_t I_b +_t IC_{t_0+I_b}$, $t >_t t_i$ and $IC_{t_0+I_b}$ is the time for executing the statement C at time $t_0 +_t I_b$, where I_b is the time for evaluating the Boolean expression b. A semantic rule for finishing the **while** loop is as follows:

$$\frac{@p_0 \cdot t_0[\![b, r_{t_0} \cap A^{p_0}_{t_0}]\!] \overset{\text{eval}}{\rightsquigarrow} @p_0\ddot{\mu} \cdot t_0 +_t I_b[\langle 0, r_{t_0+I_b}\rangle]}{@p_0 \cdot t_0[\![\textbf{while } b \textbf{ do } C, r_{t_0} \cap A^{p_0}_{t_0}]\!] \overset{\text{exec}}{\rightsquigarrow} @p_0\ddot{\mu} \cdot t_0 +_t I_b[r_{t_0+I_b} \cap A^{p_0}_{t_0}]}$$

Once two or more agents are at the same coordinate, they usually communicate locally. To provide local communication, every agent has its input queue, which is part of the state. As briefly defined in the algebra signature, $\Delta : S \times Loc \times Val \longrightarrow S$ updates the queue with a value of type Val, given a state in S and location in Loc. Moreover, $\chi : S \longrightarrow S$ removes the first element from the agent queue by receiving a state and producing another one. $\xi : S \longrightarrow \mathcal{A}^R \times S$ informs the first element of the agent queue (removing it), which is a pair that contains the value and the new state. As before, $\gamma : Var \longrightarrow Loc$ is the function that, given a variable, locates its storage. Thus, here I am not concerned with delays or faults:

$$\frac{\begin{array}{c} \xi@p_0 \cdot t_0[r_{t_0} \cap A^{p_0}_{t_0}] = \langle u, r_{t_i}\rangle \quad u \neq \textbf{\textit{uu}} \quad \xi@p_0\ddot{\mu} \cdot t_i[r_{t_i} \cap A^{p_0}_{t_i}] = \langle v, r_{t_j}\rangle \\ @p_0\ddot{\mu}\ddot{\mu} \cdot t_i +_t I2_r[\Delta(\chi\Delta(\chi r_{t_j}, \gamma par, v), \gamma V, u) = r_t] \end{array}}{@p_0 \cdot t_0[\![\textbf{read to } V, r_{t_0} \cap A^{p_0}_{t_0}]\!] \overset{\text{exec}}{\rightsquigarrow} @p_0\ddot{\mu}\ddot{\mu}\ddot{\mu} \cdot t[r_t \cap A^{p_0}_{t}]}$$

where *par*, which stands for *partner*, is a system variable that informs the sender id ϕ_{par}, $t =_t t_i +_t I2_r$ and $I_r =_t I1_r +_t I2_r$ is the time for interpretation of the **read** statement, composed of these two parts, and $t_i \geq_t t_0 +_t I1_r$. $I1_r$ is the interval of time from t_0 until obtaining $\langle u, r_{t_i}\rangle$, while $I2_r$ indicates the interval of time from t_i until receiving $\langle v, r_{t_j}\rangle$.

If the queue is empty, the evaluation results in $\textbf{\textit{uu}}$, and there is no modification in the state of the queue:

$$\frac{\xi @p_0 \cdot t_0[r_{t_0} \cap A^{p_0}_{t_0}] = \langle \boldsymbol{uu}, r_{t_i} \rangle \quad @p_0 \ddot{\mu} \cdot t_1[\Delta(r_{t_0}, \gamma V, \boldsymbol{uu}) = r_t]}{@p_0 \cdot t_0[\![\textbf{read to } V, r_{t_0} \cap A^{p_0}_{t_0}]\!] \overset{\text{exec}}{\rightsquigarrow} @p_0 \ddot{\mu}\ddot{\mu} \cdot t[r_t \cap A^{p_0}_t]}$$

where $t_0 <_t t_1 <_t t$.

Given $\eta : S \times \mathcal{A}^R \longrightarrow S$ from the signature, which inserts a value in $\mathcal{A}^R$ in the queue, the opposite operation is **write**, which writes the real number stored in the variable denoted by a, to the channel denoted by h, with the following semantics:

$$\frac{@p_0 \cdot t_0[\![a, r_{t_0} \cap A^{p_0}_{t_0}]\!] \overset{\text{eval}}{\rightsquigarrow} @p_0 \ddot{\mu} \cdot t_0 +_t I_a[u] \quad @p_0 \ddot{\mu} \cdot t_0 +_t I_a[\eta(\eta(r^h_{t_0+I_a}, u), c) = r^h_t]}{@p_0 \cdot t_0[\![\textbf{write to channel } h \textbf{ from } a, r_{t_0} \cap A^{p_0}_{t_0}]\!] \overset{\text{exec}}{\rightsquigarrow} @p_0 \ddot{\mu}\ddot{\mu} \cdot t[r_{t_0}] \;\Downarrow\; @p_0 \cdot t[r^h_t]}$$

where r^h_t is the state r of some computation π_h at time t; and c is the Id of the computation, which corresponds to ϕ_{par}, i.e. the sender Id, which is also used in the **read** statement. Therefore, the **write** statement writes in the receiver's queue the value in $\mathcal{A}^R$ and c in this order. As stated in the corresponding rules, the **read** statement obtains these elements in the same order. As well as the $\ddot{\mu}$ shift, code mobility and local communication, an additional facility for mobile agents is remote communication among agents. To define remote communication formally, I first ought to set the following rules:

- $\kappa \in SObj$ at $\langle x, y, z \rangle$ is visible by all agents if there is no agent at $\langle x, y, z \rangle$;
- $\kappa \in SObj$ at $\langle x, y, z \rangle$ is not visible by any agent at a different point if there is some agent at $\langle x, y, z \rangle$. In this case, any attempt to access this value results in $\boldsymbol{uu}$.
- $\kappa \in SObj$ at $\langle x, y, z \rangle$ is visible by agents at $\langle x, y, z \rangle$. In this case, I say that κ is local to those agents. And, since $\kappa \equiv \langle v, Y \rangle$, for all r in Y, v is visible by r.

In this model, to see the content of a point $\in \mathbb{R}$ at $\langle x, y, z \rangle$, agents execute the statement **view**$\langle x, y, z \rangle$ **time** q, whose semantics in the @-logic is:

$$\frac{\begin{array}{c} @p_0 \cdot t_0[p \neq_s p_0 \wedge \Theta @p \cdot t_1[r^p] = \langle u, \emptyset \rangle \wedge u \in \mathbb{R}] \\ @p_0 \ddot{\mu} \cdot t_f[t_f -_t t_0 <_t q \wedge \neg f\pi(p, t_f)] \end{array}}{@p_0 \cdot t_0[\![\textbf{view } p \textbf{ time } q, r^{p_0} \cap A^{p_0}_{t_0}]\!] \overset{\text{eval}}{\rightsquigarrow} @p_0 \ddot{\mu}\ddot{\mu} \cdot t_f[\langle u, r^{p_0} \rangle]} \tag{10}$$

where

$$t \geq_t t_1 \geq_t t_0 +_t \Psi +_t \Delta l_1 +_t \Delta s_1 +_t I_i \tag{11}$$

and

$$t_f \geq_t t_1 +_t \Psi +_t \Delta l_2 +_t \Delta s_2 \tag{12}$$

where I_i is the time for interpreting the **view** statement. For remote attempt to see a value that is local to other agents, there is another rule:

$$\frac{\begin{array}{c} @p_0 \cdot t[p \neq_s p_0 \wedge \Theta @p \cdot t_1[r^p] = \langle u, Y \rangle \wedge Y \neq \emptyset] \\ @p_0 \ddot{\mu} \cdot t_f[t_f -_t t_0 <_t q \wedge \neg f\pi(p, t_f)] \end{array}}{@p_0 \cdot t_0[\![\textbf{view } p \textbf{ time } q, r^{p_0} \cap A^{p_0}_{t_0}]\!] \overset{\text{eval}}{\rightsquigarrow} @p_0 \ddot{\mu}\ddot{\mu} \cdot t_f[\langle \boldsymbol{uu}, r^{p_0} \rangle]} \tag{13}$$

That is, the value in the problem domain is not accessible and, therefore, $\boldsymbol{uu}$ is provided instead.

For local access in permitted time, let $\{a_1, ..., a_n\} \subset \triangle$, and c is the Id of the agent who is currently computing. Thus, the semantics is as follows:

$$\frac{@p_0 \cdot t_1 +_t \Delta t_1[\Theta @p \cdot t_1[r^p] = \langle u, \{c, a_1, ..., a_n\}\rangle \wedge \Delta t +_t I_i <_t q]}{@p_0 \cdot t_0[\![\textbf{view } p \textbf{ time } q, r^{p_0} \cap A_{t_0}^{p_0}]\!] \overset{\text{eval}}{\rightsquigarrow} @p\ddot{\mu} \cdot t_0 +_t \Delta t +_t I_i[\langle u, r^{p_0}\rangle]} \tag{14}$$

where $t_1 \geq_t t_0$. The semantics for the timeout situation during the **view** request follows the next rule:

$$\frac{@p_0\ddot{\mu} \cdot t_f[t_f -_t t_0 =_t q \wedge f\pi(p, t_f)]}{@p_0 \cdot t_0[\![\textbf{view } p \textbf{ time } q, r_{t_0}^{p_0} \cap A_{t_0}^{p_0}]\!] \overset{\text{eval}}{\rightsquigarrow} @p_0\ddot{\mu} \cdot t_f[\langle \textbf{\textit{uu}}, r_{t_f}^{p_0}\rangle]} \tag{15}$$

where t_f is constrained in accordance with the expression 12.

In this model, to provide some form of communication, agents can throw a real number at any place in space with timeout q, and this operation takes some time to be completed, as described below. A place occupies one point in $\mathbb{E}^3$. The value $\textbf{\textit{uu}}$ can also be thrown and indeed can be part of programming languages constructs as introduced in.

The operational semantics of the **throw** instruction is as follows:

$$\frac{\begin{array}{c}@p_0 \cdot t_0[\![a, r^{p_0} \cap A_{t_0}^{p_0}]\!] \overset{\text{eval}}{\rightsquigarrow} @p_0\ddot{\mu} \cdot t_0 +_t I_a[\langle v, r^{p_0}\rangle] \\ @p_0 \cdot t[t -_t t_0 <_t q \wedge \neg f\pi(p, t)] \quad @p_0\ddot{\mu} \cdot t[\Theta @p \cdot t[r^p] = \langle u, \emptyset\rangle]\end{array}}{@p_0 \cdot t_0[\![\textbf{throw } a \textbf{ to } p \textbf{ time } q, r^{p_0} \cap A_{t_0}^{p_0}]\!] \overset{\text{exec}}{\rightsquigarrow} @p_0\ddot{\mu} \cdot t[r^{p_0}] \downdownarrows @p \cdot t[\langle v, \emptyset\rangle]} \tag{16}$$

where

$$t \geq_t t_0 +_t \Psi +_t \Delta l +_t I_a +_t Th$$

where Th is the time for interpreting the throw instruction.

In the case that there is already some agent at $\langle x, y, z\rangle$, if an agent throws a number from another place, there is no effect. Therefore, if

$$\{a_1, a_2, ..., a_n\} \subseteq \triangle$$

is a non-empty set of agents, and $t_0 \leq_t t_1 \leq_t t$:

$$\frac{\begin{array}{c}@p_0 \cdot t_0[\![a, r^{p_0} \cap A_{t_0}^{p_0}]\!] \overset{\text{eval}}{\rightsquigarrow} @p_0\ddot{\mu} \cdot t_0 +_t I_a[\langle v, r^{p_0}\rangle] \\ @p_0 \cdot t[t -_t t_0 <_t q \wedge \neg f\pi(p, t)] \quad @p_0\ddot{\mu} \cdot t[\Theta @p \cdot t_1[r^p] = \langle u, \{a_1, a_2, ..., a_n\}\rangle]\end{array}}{\begin{array}{c}@p_0 \cdot t_0[\![\textbf{throw } a \textbf{ to } p \textbf{ time } q, r^{p_0} \cap A_{t_0}^{p_0}]\!] \overset{\text{exec}}{\rightsquigarrow} \\ @p_0\ddot{\mu} \cdot t[r^{p_0}] \downdownarrows @p \cdot t[\langle u, \{a_1, a_2, ..., a_n\}\rangle]\end{array}} \tag{17}$$

The timeout condition for the **throw** statement has the following rule:

$$\frac{\begin{array}{c}@p_0 \cdot t_0[r^{p_0} \subseteq A_{t_0}] \quad @p_0\ddot{\mu} \cdot t[t -_t t_0 =_t q] \quad @p_0 \cdot t_0[r^{p_0} \subseteq A_t] \\ @p_0 \cdot t[f\pi(p, t)] \quad @p_0\ddot{\mu} \cdot t[\Theta @p \cdot t_0[r^p] = \langle u, \emptyset\rangle]\end{array}}{\begin{array}{c}@p_0 \cdot t_0[\![\textbf{throw } a \textbf{ to } p \textbf{ time } q, r^{p_0} \cap A_{t_0}]\!] \overset{\text{exec}}{\rightsquigarrow} \\ @p_0\ddot{\mu} \cdot t[r^{p_0} \cap A_t] \downdownarrows @p \cdot t[\langle u, \emptyset\rangle]\end{array}} \tag{18}$$

Notice that real numbers can contain the codification of some finite mobile agent. This means that the **throw** statement is able to model this model. I formalize the sequence of statements:

$$\frac{@p_0 \cdot t_0[\![C_1, r_{t_0} \cap A^{p_0}_{t_0}]\!] \overset{\text{exec}}{\rightsquigarrow} @p_i \cdot t_i[r_{t_i} \cap A^{p_i}_{t_i}] \quad @p_i \cdot t_i[\![C_2, r_{t_i} \cap A^{p_i}_{t_i}]\!] \overset{\text{exec}}{\rightsquigarrow} @p \cdot t[r_t \cap A^p_t]}{@p_0 \cdot t_0[\![C_1; C_2, r_{t_0} \cap A^{p_0}_{t_0}]\!] \overset{\text{exec}}{\rightsquigarrow} @p \cdot t[r_t \cap A^p_t]}$$

In spite of the greater generality of the present model, in the next section, I shall present an example that demands a more powerful notion of computation in comparison to the one I have presented up to this point. Following this, I shall complement the present model with the **SpreadOut** operation, which is novel in computer science, and whose semantics is also formalized.

The crucial aspect here is that, unlike the **flyto** statement, this new proactive move is broadcasting publicly, and without any address specification for the destination. The only condition for the continuation is the agent be accepted by the destination, and for which I should set the predicate *receptive*, which states that the destination is receptive to the present computation. In this way, *rec(r)* indicates that the surrounding ambient is receptive to the state r, and now the **SpreadOut** operation can be conceived and included in the present model. Because the operation does not require any operand, here I use the syntax **SpreadOut**. The semantics is as follows:

$$\frac{\forall p \in \mathcal{U} \quad @p_0 \cdot t_0[r \subseteq A^{p_0}_{t_0}] \quad @p \cdot t_i[rec(r) \in A^p_{t_i}] \quad @p \cdot t_i[r \not\subseteq A^p_{t_i}] \quad @p \cdot t[t =_t t_0 +_t q \wedge \neg f\pi(p,t) \wedge r \subseteq A^p_t]}{\begin{array}{c} @p_0 \cdot t_0[\![\textbf{SpreadOut}\ q, r \cap A^{p_0}_{t_0}]\!] \overset{\text{exec}}{\rightsquigarrow} \\ @p_0 \ddot{\mu} \cdot t_0 +_t I_s[r \cap A^{p_0}_{t_0 +_t I_s}] \Downarrow @p \ddot{\mu} \cdot t[r \cap A^p_t \setminus \{rec(r)\}] \end{array}}$$

and, finally, one additional rule for the case that there is no receptions:

$$\frac{\forall p \in \mathcal{U} \quad @p_0 \cdot t_0[r^{p_0} \subseteq A^{p_0}_{t_0}] \quad @p \cdot t_i[(rec(r^{p_0}) \notin A^p_{t_i}) \vee r^{p_0} \subseteq A^p_{t_i}]}{@p_0 \cdot t_0[\![\textbf{SpreadOut}\ q, r^{p_0} \cap A^{p_0}_{t_0}]\!] \overset{\text{exec}}{\rightsquigarrow} @p_0 \ddot{\mu} \cdot t_0 +_t I_s[r^{p_0} \cap A^{p_0}_{t_0 +_t I_s}]}$$

where both $t =_t t_i +_t \Delta t$ and $t_i \geq_t t_0 +_t \Psi +_t \Delta l +_t \Delta s +_t I_s$ hold for every p, and I_s is the time for interpreting the **SpreadOut** statement. Notice that this operation is asynchronous.

It is *not* impossible to have timeout for the **SpreadOut** operation, and the corresponding rule is as follows:

$$\frac{\forall p \in \mathcal{U} \quad @p_0 \cdot t_0[r^{p_0} \subseteq A^{p_0}_{t_0}] \quad @p \cdot t_0 +_t q[f\pi(p, t_0 +_t q) \vee r^{p_0} \subseteq A^p_{t_0 +_t q}]}{@p_0 \cdot t_0[\![\textbf{SpreadOut}\ q, r^{p_0} \cap A^{p_0}_{t_0}]\!] \overset{\text{exec}}{\rightsquigarrow} @p_0 \ddot{\mu} \cdot t_0 +_t I_s[r^{p_0} \cap A^{p_0}_{t_0 +_t I_s}]}$$

Such a move towards the user interface is another example of centralization.

The idea presented here also attempts to illustrate the power of the combination of mobile agents with broadcasting, which might popularize parallel computation and provide an evolutionary model. The sequence of moves generated randomly provides mutation with respect to its prefix. As well as efficiency concerns and no need for providing the destination address as opposed to, for example, the Internet, there are other important issues in this example: like sperm cells in a female animal, mobile agents can *compete* for the best solution, which leads designers to questions in *ethics*. "What should agents be allowed to do?" This is an open question. Ethics for mobile agents on a global environment may be even more complicated because agents often transcend boundaries

of countries. "How can one distinguish an agent from a virus?" This is another question.

This application exemplifies a model of computing to solve problems that include the possibility of combinatorial explosion. One can provide an interesting algorithm to find a mobile agent on a network. The use of a media such as radio[2] together with mobile agents is an easy alternative to solve the same problem.

Another issue is how should a mobile agent system prevent agents from multiplying in an uncontrolled manner? What is the reasonable cost, overhead and burden of such an operation? To date, there has not been a technology that guarantees total security for a host that receives a broadcasted mobile agent, but any mobile code system that support such operations should see the network as public. A similar problem is to access WWW hosts in parallel by software modules, while this idea might be desirable from some people's point of view but could also cause traffic jams if performed by programs too often. Therefore, it is clear that there is an open door to philosophy in computer science, while AI gets closer to the foundations of computer science[30].

8 Conclusion

Among other conclusions, I discussed mobility and a more general notion of computation that includes physical, psychological and philosophical factors[30]. I propose four forms of mobility in this physical and simplified model.

Up to some level of abstraction, a simple and different model in $\mathbb{E}^3$ has been presented here by defining a machine together with a somewhat minimal set of constructs that complements the traditional notion of computation, as well as an operational semantics for these constructs.

For this purpose, I have introduced one statement, namely **SpreadOut**, and have written its operational semantics, and included the operation in the present model. I shall call the corresponding paradigm *broadcasting mobility*. There might be other forms of broadcasting mobility in the near future. The Internet is used by people from different places from different backgrounds as philosophy increases its importance for computer science. Furthermore, this change from mathematics to also include other studies also opens a door to changes in scientific methods[30]. Science in computer science has a broader sense.

References

[1] J. Allen. Time and time again: The many ways to represent time. *International Journal of Intelligent Systems*, 6(4):341–355, July 1991.

[2] J. Allen and G. Ferguson. Actions and events in interval temporal logic. *Journal of Logic and Computation*, 4(5), 1994.

[3] J. Allen and J. Hendler, editors. *Readings in Planning*. Representation and Reasoning. Morgan Kaufmann, San Mateo, California, 1990.

[2] the use of radio is figurative, to stand for any multicast media, e.g. a display visible to all agents would suffice.

[4] R. M. Amadio. On modelling mobility. *Theoretical Computer Science*, 240(1):147–176, June 2000.

[5] J.-M. Autebert, J. Berstel, and L. Boasson. *Hanbook of Formal Languages*, volume 1, chapter Context-Free Languages and Pushdown Automata, pages 111–174. Springer-Verlag, 1997.

[6] F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998.

[7] H. Balen. *Distributed object architectures with CORBA.* Cambridge University Press, 1999.

[8] C. Berline. From computation to foundations via functions and application: The λ-calculus and its webbed models. *Theoretical Computer Science*, 249(1):81–161, October 2000.

[9] C. Bettini, S. Jajodia, and S. X. Wang. *Time Granularities in Databases, Data Mining and Temporal Reasoning.* Springer-Verlag, 2000.

[10] L. F. Bic, M. Fukuda, and M. B. Dillencourt. Distributed computing using autonomous objects. *IEEE Computer*, Aug. 1996.

[11] G. S. Boolos and R. C. Jeffrey. *Computability and Logic.* Cambridge University Press, third edition, 1989.

[12] M. Broy. Refinement of time. *Theoretical Computer Science*, 253(1):3–26, February 2001.

[13] G. Bruns. *Distributed Systems Analysis with CCS.* Prentice-Hall International Series in Computer Science. Prentice-Hall Europe, 1997.

[14] M. Bugliesi and G. Castagna. Secure safe ambients. In *Proceedings of the POPL 2001, XXVIII ACM SIGPLAN - SIGACT, Symposium on Principles of Programming Languages*, pages 222–235. ACM SIGPLAN, SIGACT, 2001. Also SIGPLAN Notices 36(3):222–235.

[15] L. Cardelli. Global computation. *ACM Computing Surveys*, 28A(4), 1996.

[16] L. Cardelli. Mobile computation. In *Mobile Object Systems: Towards the Programmable Internet*, pages 3–6. Springer-Verlag, Apr. 1997. Lecture Notes in Computer Science No. 1222.

[17] L. Cardelli. *Mobile Object Systems*, chapter Mobile Computation. Number 1222 in Lecture Notes in Computer Science. Springer-Verlag, Linz, Austria, 1997.

[18] L. Cardelli and A. D. Gordon. *Foundations of Software Science and Computational Structures*, volume 1378 of *Lecture Notices in Computer Science*, chapter Mobile Ambients, pages 140–155. Springer-Verlag, 1998. Also Proceedings of FoSSaCS'98.

[19] L. Cardelli and A. D. Gordon. Mobile ambients. *Theoretical Computer Science*, 240(1):177–213, June 2000.

[20] T. S. Champlin. *Reflexive Paradoxes*. Routledge, 1988.

[21] D. Chess, C. Harrison, and A. Kershenbaum. Mobile agents: Are they a good idea? – update. In *Mobile Object Systems: Towards the Programmable Internet*, pages 46–48. Springer-Verlag, Apr. 1997. Lecture Notes in Computer Science No. 1222.

[22] A. Church. *Introduction to Mathematical Logic*. Princeton Mathematical series. Princeton University Press, 1956. Tenth printing (1996) for the Princeton Landmarks in Mathematics and Physics series.

[23] R. Connor and K. Sibson. Paradigms for global computation - an overview of the hippo project. In *Proceedings of Computer Society International Conference on Computer Languages*, Chicago, 1998. IEEE.

[24] B. J. Copeland, editor. *Logic and Reality: essays on the legacy of Arthur Prior*. Oxford University Press, 1996.

[25] R. Cori and D. Lascar. *Mathematical Logic: a course with exercises*, volume 2. Oxford University Press, 2001. original in French.

[26] G. M. Corp. *Odyssey White Paper*, 1998.

[27] G. Cugola, C. Ghezzi, G. P. Picco, and G. Vigna. Analyzing mobile code languages. In *Mobile Object Systems: Towards the Programmable Internet*, pages 93–110. Springer-Verlag, Apr. 1997. Lecture Notes in Computer Science No. 1222.

[28] J. W. Dauben. *Georg Cantor: his mathematics and philosophy*. Harvard University Press, 1979.

[29] J. R. Durbin. *Modern Algebra: an introduction*. John Wiley & Sons, Inc., fourth edition, 2000.

[30] U. Ferreira. On the foundations of computing science. In D. L. Hicks, editor, *Proceedings of the Metainformatics Symposium MIS'03*, number 3002 in Lecture Notes in Computer Science, pages 46–65. Springer, September 2003, published in 2004.

[31] C. Fields. *Machines and Thought: The Legacy of Alan Turing*, volume 1 of *Mind Association ocasional series*, chapter Mensurement and Computational Description, pages 165–177. Oxford University Press, 1996.

[32] C. Fournet and G. Gonthier. The reflexive chemical abstract machine and the join-calculus. In *Proceedings of 23rd ACM Symposium on Principles of Programming Languages*, January 1996.

[33] C. Fournet, G. Gonthier, J.-J. Lévy, L. Maranget, and D. Rémy. A calculus of mobile agents. In *7th International Conference on Concurrency Theory (CONCUR'96)*, pages 406–421, Pisa, Italy, August 1996. Springer-Verlag. LNCS 1119.

[34] A. Fuggetta, G. P. Picco, and G. Vigna. Understanding code mobility. *IEEE Transactions on Software Engineering*, 24(5), May 1998.

[35] D. M. Gabbay, I. Hodkinson, and M. Reynolds, editors. *Temporal Logic: mathematical foundations and computational aspects*, volume 1. Oxford University Press Inc., New York, 1994.

[36] D. Gentner, K. J. Holyoak, and B. N. Kokinov, editors. *The Analogical Mind: perspectives from cognitive science*. The MIT Press, 2001.

[37] J.-Y. Girard, P. Taylor, and Y. Lafont. *Proofs and Types*. Cambridge University Press, 1993.

[38] A. D. Gordon. *Functional Programming and Input/Output*. Distinguished Dissertations in Computer Science. Cambridge University Press, 1994.

[39] C. A. Gunter. *Semantics of Programming Languages: structures and techniques*. Foundations of Computing Series. The MIT Press, 1992.

[40] J. Honkala. On parikh slender context-free languages. *Theoretical Computer Science*, 255(1–2):667–677, March 2001.

[41] M. N. Huhns and M. P. Singh, editors. *Readings in Agents*. Morgan Kaufmann, San Francisco, California, 1997.

[42] O. Ibidapo-Obe, O. S. Asaolu, and A. B. Badiru. Generalized solutions of the pursuit problem in three-dimensional euclidean space. *Applied Mathematics and Computation*, 119(1):35–45, March 2001.

[43] R. Incitti. The growth function of context-free languages. *Theoretical Computer Science*, 255(1–2):601–605, March 2001.

[44] N. D. Jones. *Computability and Complexity: from a programming perspective*. Foundations of Computing. The MIT Press, 1997.

[45] C. Jung et al. *Man and His Symbols*. Adus Books Ltda, Pan MacMillan, 20 New Wharf Road, London N1 9RR, 1964. Conceived and Edited by Carl Jung. Newer edition published by Picador, in 1978.

[46] I. Kant. *Logic*. Bobbs-Merrill, Indianapolis, 1974. Original in German Logik. Translation with an introduction by Robert S Hartman and Wolfgang Schwarz.

[47] A. J. P. Kenny. *A Brief History of Western Philosophy*. Blackwell Publishers, 1998.

[48] B. Kirkerud. *Programming Language Semantics*. International Thomson Computer Press, 1997.

[49] B. Kosko. *Fuzzy Thinking: The New Science of Fuzzy Logic*. HarperCollinsPublishers, Flamingo, 1994.

[50] K. B. Krauskopt and A. Beiser. *The Physical Universe*. McGraw-Hill Companies, Inc, eighth edition, 1997.

[51] L. Lamport and N. Lynch. *Handbook of Theoretical Computer Science*, volume B Formal Models and Semantics, chapter 18 Distributed Computing: Models and Methods, pages 1157–1199. The MIT Press/Elsevier, 1990.

[52] D. Langford, editor. *Internet Ethics*. MacMillan Press Ltd, Printed and Bound in Great Britain by Antony Rowe Ltd, Chippenham, Wiltshire, 2000.

[53] Z. Luo. *Computation and Reasoning: a type theory for computer science*, volume 11 of *International Series of Monographs on Computer Science, edited by Gabbay, Hopcroft, Plotkin, Schwartz, Scott, Vuillemin and Galil*. Oxford Science Publications, 1994.

[54] K. Meinke and J. V. Tucker. *Handbook of Logic in Computer Science*, volume 1: Mathematical Structures, chapter Universal Algebra, pages 189–411. Oxford University Press, 1992.

[55] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes. part I and II. *Information and Computation*, 1(100), 1992.

[56] D. S. Milojíčić, F. Douglis, Y. Paindaveine, R. Wheeler, and S. Zhou. Process migration. *ACM Computing Surveys*, 32(3):241–299, September 2000.

[57] J. C. Mitchell. *Foundations for Programming Languages*. Foundations of Computing. The MIT Press, 1996.

[58] F. Monin and M. Simonot. An ordinal measure based procedure for termination of functions. *Theoretical Computer Science*, 254(1–2):63–94, March 2001.

[59] H. R. Nielson and F. Nielson. *Semantics with Applications: a formal introduction*. John Wiley & Sons, 1993.

[60] ObjectSpace Corp. *Voyager White Paper*, 1998.

[61] G. P. Picco. Mobile agents: an introduction. *Microprocessors and Microsystems*, 25(2):65–74, April 2001.

[62] S. Rajasekaran and P. Pardalos, editors. *Mobile Networks and Computing: DIMACS Workshop 99*, volume 52 of *DIMACS series in discrete mathematics and theoretical computer science*. American Mathematical Society, March 2000.

[63] V. J. Rayward-Smith. *A First Course in Computability*. McGRAW-HILL Book Company Europe, 1986.

[64] R. Rodriguez and F. Anger. *Logic and Reality: essays on the legacy of Arthur Prior*, chapter Prior's Temporal Legacy in Computer Science, pages 89–105. Oxford University Press, 1996.

[65] K. Rothermel and F. Hohl, editors. *Mobile Agents: Second International Workshop / MA'98*, volume 1477 of *Lecture Notes in Computer Science*. Springer, Stuttgart, September 1998. Proceedings.

[66] G. Rozenberg and A. Salomaa, editors. *Handbook of Formal Languages*, volume 1. Springer-Verlag, 1997.

[67] Y. V. Sachkov. *International Congress of Logic, Methodology, and Philosophy of Science (6th : 1979: Hannover): Logic, Methodology, and Philosophy of Science*, volume 104 of *Studies in Logic and the Foundations of Mathematics*, chapter Foundations and Philosophy of the Physical Sciences, Probability in Classical and Quantum Physics, pages 441–447. PWN-Polish Scientific Publishers-Warszawa and North Holland Publishing Company, 1982.

[68] P. Sewell, P. T. Wojciechowski, and B. C. Pierce. Location-independent communication for mobile agents: a two-level architecture. In *Internet Programming Languages*, number 1686 in Lecture Notes in Computer Science, pages 1–31. Springer-Verlag, 1998.

[69] M. Sipser. *Introduction to the Theory of Computation*. PWS Publishing Company, 1997.

[70] K. Slonneger and B. L. Kurtz. *Formal Syntax and Semantics of Programming Languages: a laboratory based approach*. Addison-Wesley Publishing Company, 1995.

[71] R. G. Taylor. *Models of Computation and Formal Languages*. Oxford University Press, 1998.

[72] D. V. Thanh, S. Steensen, and J. A. Audestad. Mobility management and roaming with mobile agents. In *Mobile and Wireless Communications Networks*, number 1818 in Lecture Notes in Computer Science, pages 123–137. Springer-Verlag Berlin Heidelberg, 2000.

[73] J. V. Tucker and J. I. Zucker. *Handbook of Logic in Computer Science*, volume 5: Logic and Algebraic Methods, chapter Computable Functions and Semicomputable Sets on Many-Sorted Algebras, pages 317–523. Oxford University Press, 2000.

[74] A. Unyapoth and P. Sewell. Nomadic pict: Correct communication infrastructure for mobile computation. In *Proceedings of the POPL 2001, XXVIII ACM SIGPLAN - SIGACT, Symposium on Principles of Programming Languages*, pages 116–127. ACM SIGPLAN, SIGACT, 2001. Also SIGPLAN Notices 36(3):116–127.

[75] J. Vitek and G. Castagna. Seal: A framework for secure mobile computation. In *Internet Programming Languages*, number 1686 in Lecture Notes in Computer Science, pages 47–77. Springer-Verlag, 1998.

[76] P. Wadler. How to declare an imperative. *ACM Computing Surveys*, 29(3):240–263, September 1997.

[77] J. Waldo, G. Wyant, A. Wollrath, and S. Kendall. A note on distributed computing. In *Mobile Object Systems: Towards the Programmable Internet*, pages 49–64. Springer-Verlag, Apr. 1997. Lecture Notes in Computer Science No. 1222. Also published as Sun Microsystems Laborators Technical Report TR-94-29.

[78] G. Winskel. *The Formal Semantics of Programming Languages: an introduction*. The MIT Press, fourth edition, 1997.

[79] G. Winskel. *The Formal Semantics of Programming Languages: An Introduction*. Foundations of Computing. The MIT Press, 1997.

[80] A. Yasuhara. *Recursive Function and Logic*. Academic Press, Inc, 1971.