Form Method Syst Des (2007) 31:1–33
DOI 10.1007/s10703-006-0024-z

Refinement preserving approximations for the design and verification of heterogeneous systems

Roberto Passerone · Jerry R. Burch · Alberto L. Sangiovanni-Vincentelli

Received: 26 July 2005 / Accepted: 12 September 2006 /
Published online: 12 October 2006

Abstract Embedded systems are electronic devices that function in the context of a real environment, by sensing and reacting to a set of stimuli. Because of their close interaction with the environment, and to simplify their design, different parts of an embedded system are best described using different notations and different techniques. In this case, we say that the system is *heterogeneous*. We informally refer to the notation and the rules that are used to specify and verify the elements of heterogeneous systems and their collective behavior as a *model of computation*. In this paper, we consider different classes of relationships between models of computation and discuss their preservation properties with respect to the model's refinement relation and composition operator. In particular, we focus on abstraction and refinement relationships in the form of *abstract interpretations* and introduce the notion of *conservative approximation*. We show that, unlike abstract interpretations, conservative approximations preserve refinement verification results from an abstract to a concrete model while avoiding false positives. We also characterize the relationship between abstract interpretations and conservative approximations, and derive necessary and sufficient conditions to obtain a conservative approximation from a pair of abstract interpretations. In addition, we use the inverse of a conservative approximation to identify components that can be used indifferently in several models, thus enabling reuse across models of computation. The concepts described in this paper are illustrated with examples from continuous time and discrete time models of computation.

R. Passerone (✉)
Cadence Design Systems, 1995 University Ave Suite 460, Berkeley CA 94709

Present Address: Department of Information and Communication Technology, University of Trento, via Sommarive 14, 38050 Povo (TN), Italy

J. R. Burch
Synopsys, Inc., 2025 NW Cornelius Pass Rd., Hillsboro, OR 97124

A. L. Sangiovanni-Vincentelli
Department of EECS, University of California, Berkeley, Berkeley CA 94720

Keywords Refinement · Preserving · Approximation · Abstraction · Verification · Heterogeneous · Reuse · Polymorphism · Model of computation · Galois connection · Abstract interpretation · Conservative approximation · Continuous time · Discrete time · Refinement · Concretization

1 Introduction

Embedded systems are electronic devices that function in the context of a real environment, by sensing and reacting to a set of stimuli. Because of their close interaction with the environment, and to simplify their design, different parts of an embedded system are best described using different notations and different techniques. In this case, we say that the system is *heterogeneous*. For example, the model of the software application that runs on a distributed collection of nodes in a sensor network is often concerned only with the initial and final state of the behavior of a reaction. In contrast, the particular sequence of actions of the reaction could be relevant to the design of one instance of a node. Likewise, the notation employed in reasoning about a resource management subsystem is often incompatible with the handling of real time deadlines, typical of communication protocols. This form of heterogeneity is also reflected in the structure of the design teams, which increasingly consist of highly specialized groups that focus on the solution of a particular task, under the direction of the system architects.

Designers benefit from this separation. First, the system is naturally partitioned into smaller and more manageable parts. Secondly, and more importantly, designers are free to select for each subsystem the rules that are used to specify its behavior as a hierarchical collection of modules (*composition*), and to verify that such behavior conforms to a specification (*refinement verification*). These rules vary widely across different modeling domains, such as the ones outlined above. The restrictions and the intrinsic properties of these rules, which we collectively refer to as a *model of computation*, are the basis of domain specific techniques that can be used to more easily guarantee the correctness of the implementation.

While specified separately, subsystems must eventually interact to form the system behavior, and will in fact do so in the physical implementation. However, system designers are typically unwilling to wait until the final stages of the implementation to validate the system functionality and performance metrics, because the cost of fixing design and specification errors increases dramatically in the later phases of the design flow as amply documented for electronic systems, software and integrated circuits. The costs associated with late discovery of errors and, in particular, of integration errors, have risen to a point that they are no longer sustainable. To witness, consider the recent recalls by Mercedes-Benz of 1.5 million cars for problems with the braking subsystem. Consequently, the ability of the system designer to specify, manage, and verify the functionality and performance of *concurrent behaviors*, within *and across* heterogeneous boundaries, is essential. Most design methodologies that address these problems are based on the processes of abstraction and refinement, that is, of the application of maps that convert and relate different models of computation. However, crossing the boundaries between abstraction levels by abstracting and refining a specification is often not trivial. The most common pitfalls include mishandling of corner cases and inadvertently misinterpreting changes in the communication semantics. These problems arise because of the poor understanding and the lack of a precise definition of the abstraction and refinement maps used in the flow, which are therefore likely to provide little if any guarantee of satisfying a given set of constraints and specifications, without resorting to extensive simulation or tests on prototypes. However, in the face of growing complexity, this approach

will have to yield to more rigorous methods. In addition, abstraction and refinement should be designed to preserve, whenever possible, the properties of the design that have already been established. This is essential to increase the value of early, high level models and to guarantee a speedier path to implementation.

The objective of this paper is to approach the problem of abstraction and refinement from a formal standpoint, and to study and compare the preservation properties of different forms of abstraction. In particular, we study abstractions that preserve positive refinement verification results (the relation between an implementation and a specification) from an abstract modeling domain to a concrete modeling domain. This property of an abstraction is useful because, presumably, verification is more efficient at the abstract level than it is at the concrete. In this paper, we focus in particular on abstraction and refinement relationships in the form of *abstract interpretations* [8, 9] and of *conservative approximations* [3–6, 24]. Abstract interpretations are a widely used means of relating different domains of computation for the purpose of facilitating the analysis of a system. An abstract interpretation between two domains of computation consists of an abstraction function and of a concretization function that form a Galois connection. The distinguishing feature of an abstract interpretation is that the concretization of the evaluation of an expression using the operators of the abstract domain of computation is guaranteed to be an upper bound of the corresponding evaluation of the same expression using the operators of the concrete domain. Hence, a conservative evaluation can be carried out at the more abstract level, where it is potentially computationally more efficient. Refinement verification, however, is unsound: a positive refinement verification result at the abstract level does not guarantee a corresponding refinement verification result at the concrete level.

This problem is overcome by using conservative approximations. The concept of a conservative approximation in our work is derived from the one introduced by Burch [3]. Here we generalize this approach and apply it to a domain of arbitrary agents, rather than assuming that an agent is modeled by a set of executions. We also decompose the definition of conservative approximation to highlight and discuss its compositionality properties, and study its relationship with traditional notions of abstraction. Conservative approximations are closely related to abstract interpretations, as will be shown later in Section 6. We show, however, that unlike Galois connections, conservative approximations preserve refinement verification from an abstract to a concrete model while avoiding the occurrence of false positive results. This can be accomplished with conservative approximations because they employ two separate abstraction functions, one for the implementation and one for the specification. Our study also shows that this is a necessary condition for the preservation of refinement, and one that is not satisfied by a Galois connection. Conservative approximations and abstract interpretations are however strongly related. The main result of this paper is that *a pair of Galois connections can be used to construct a conservative approximation*. This result is important because it extends the rich field of abstract interpretations to refinement verification. We examine and determine the exact conditions under which this result holds.

The study of heterogeneous systems is also a central theme of both the Metropolis [2] and the Ptolemy [18] projects. In Metropolis, a system is composed of processes that communicate over media expressed in a meta-model of computation. Their combination, and their relationships, implicitly determine the interaction semantics. Because of its generality, the underlying meta-model fabrics can be used to promote reuse of diverse components. The communication media, however, must be carefully designed to resolve possible incompatibilities. Our work can be thought of as the theory base for the use of the meta-model to represent heterogeneous systems. The techniques presented in this paper, in fact, expose the relationships between the different models and help the designer build media that adapt

processes by "completing" their set of behaviors, and that therefore comply with a different interaction semantics. In addition, conservative approximations have been used to make the process of platform-based design advocated in the Metropolis project precise, and their application in this area is part of our current work [24].

Similarly, Ptolemy consists of several executable domains of computation that can be mixed in a hierarchy controlled by a global scheduler. Ptolemy does not currently provide a notion of abstraction between the different models in the system. However, an important innovative concept in the design of the Ptolemy II infrastructure is the notion of *domain polymorphism* [19]. An actor (agent) is domain polymorphic if it can be used indifferently, i.e., without modification, in several domains of computation. To check whether an actor can be used in a particular domain, the authors set up a type system based on state machines, which is used to describe the assumptions of each model and each actor relative to an abstract semantics.

Conservative approximations offer a formal way of defining a similar concept of polymorphism, even though they do not rely upon a common underlying semantics, as in the case of Ptolemy. A distinctive feature of conservative approximations is their ability to determine which parts of the models are unaffected by the application of the abstraction. This information is useful because it identifies the elements of the models that can be expressed indifferently under the interpretation of either model, without changing their meaning. Our interpretation of this notion is, however, broader than that introduced in Ptolemy II. In particular, an actor (agent) is polymorphic in our framework when it makes no assumption regarding its behavior based on information that cannot be expressed in the other model. When this is the case, reuse of subsystems can be extended across the boundaries of heterogeneous models. This leads to the notion of the *inverse* of a conservative approximation, which is a refinement map that is used to embed one model into another. The embedding provides us with an interpretation of agents across different models which is consistent with the corresponding abstraction. *An agent is polymorphic precisely when this interpretation is exact*. This has the advantage of making the process of abstraction and refinement of an agent explicit. Elements that do not fall in the range of the inverse can only be approximated by the other model. Nonetheless, we show how the inverse can be used in conjunction with the abstraction maps to construct operators that add or remove behaviors to ensure compatibility across model boundaries.

To simplify the exposition, and to be precise about the relation of our approach with previous literature, other forms of abstraction are discussed at the end of this paper in Section 8. In all cases, the specific abstraction is shown to be either an instance of an abstract interpretation (and is therefore unsound for refinement verification), or is a particular case of conservative approximations. The rest of the paper is structured as follows. Section 2 introduces various forms of abstractions, including Galois connections and conservative approximations, and compares their preservation properties relative to refinement verification. Section 3 presents an extensive example of the use of conservative approximations for an abstraction from a continuous time to a discrete time model of computation. Section 4 is devoted to examining the compositionality properties of the abstractions, relative to the operators of the models. Then, we introduce refinement functions in Section 5. Section 6 presents the main result of this paper, an exact characterization of the relationships between the different kinds of abstractions introduced here. Finally, we use both abstraction and refinement maps to develop a notion of polymorphism for the elements of different models of computation, and we explore ways to translate agents from one model to another. To avoid interrupting the flow

of the paper, the proofs of the main results are included at the end in Appendix A. The other, more technical, proofs are omitted. The interested reader is referred to [24] for more details.

2 Preservation properties

We have made the case in the introduction that it is essential in a heterogeneous design methodology to understand how different abstraction levels, or models, relate to each other. As we have noted, this can be done by building certain relationships between the models. These relationships are usually classified as abstractions (going from the more concrete to the abstract) or refinements (going from the more abstract to the concrete). After presenting preliminary material, this section reviews several forms of abstraction (and their corresponding refinements) and compares their ability to preserve certain properties of interest. In particular, we show that monotonic functions are insufficient to preserve certain refinement relationships, and that Galois connections can be used for that same purpose only in restricted circumstances. Conservative approximations are introduced to overcome these problems.

2.1 Preliminaries

Before we can investigate the notion of an abstraction, we must provide a way to describe its domain and range, namely the models of computation. In general, an abstraction transforms a block of computation in one model into a block of computation in another model. For example, it may transform a module written in a discrete event language (such as Verilog or VHDL) into a transaction level module that ignores the precise time at which events occur, such as a dataflow language. We therefore represent models of computation at the granularity of the module, or block. In other words, a model of computation is simply the set of blocks that can be expressed in the model. For instance, we represent a model of computation based on finite state machines as the set of finite state machines, or a dataflow model of computation as the set of dataflow actors. However, the representation of the blocks need not be in the form of a programming language. In fact, to simplify the task of defining abstraction functions, we typically represent blocks as the *set of behaviors* that they can exhibit. The nature of these behaviors obviously depends on the particular model of computation: for instance, they may consist of sequences of values (as in the case of synchronous models), functions of real variables (for more accurate continuous time models), or simply sets of values representing certain performance metrics (power models, constraints). Because we use behaviors, we will refer to blocks in any model of computation generically as *agents*, and they will be denoted by the letters p and q. Nonetheless, our discussion about abstraction maps below is independent of the particular way the agents are represented.

Agents in a model of computation do not exist in isolation. In general, they can be combined with the operation of parallel composition (denoted by the symbol $\|$) to yield a new agent in the *same* model of computation. The new agent combines the behaviors of the original components. For example, finite state machines can be combined by taking their product, while discrete event processes are combined by synchronizing their respective event queues. Several different forms of composition are possible, and each characterizes a different communication semantics for a model of computation. Alternatively, one can use a fixed communication scheme and model different communication semantics explicitly by employing appropriate "communication" agents to mediate the transactions between "computation" agents (this is, for example, the approach taken in the Metropolis meta-model [2], where *communication media* are used to define the interaction semantics).

Other operations on the model can also be used to hide or expose internal details of the components, or to instantiate them to form a system. We refer to the first as *projection* (and its inverse), and denote it by the operator *proj*. Instantiation, on the other hand, can be seen as changing the name of formal parameters into actual parameters. We therefore refer to it as a *renaming* operation, and denote it by the operator *rename*.

Different means can often be used to achieve the same goal. Likewise, agents with different behavior may sometimes yield the same result when applied to a particular context. In particular, if an agent p can always be replaced for an agent q in any context without materially changing the outcome of the composition, then we say that p *refines* q. In the rest of this paper we use the symbol $\preceq$ to denote this refinement relationship, and we write $p \preceq q$ whenever p refines q. We also refer to q as a *specification*, and to p as an *implementation* of q. Notice that the word "refinement" denotes both the refinement relationship within each model of computation, *and* the refinement function that maps a more abstract model into a more concrete one. This is unfortunate, but is part of the common usage. In particular, the symbol $\preceq$ will be used here exclusively to denote the refinement relationship between agents that belong to the *same* model of computation.

The following definition summarizes the above discussion.

Definition 2.1. A *model of computation* $\mathcal{Q}$ consists of a set of agents D, certain operators on D, such as parallel composition, projection and renaming, and a partial order relationship $\preceq$ on D, called refinement.

In informal discussions, and to simplify our notation, we will often use the symbol $\mathcal{Q}$ to denote its set of agents D, thus ignoring the distinction between a model and its elements. The context will make clear what is meant. We also restrict our attention to models that are partially ordered according to the refinement relationship. Refinement, however, is often not antisymmetric, yielding a preorder instead of a partial order. Our theory is easily extended to this case by simply considering the equivalence classes induced by the preorder [24]. This restriction is used here to simplify our notation and arguments.

2.2 Refinement preserving abstractions

The choice of levels of abstraction, or models, in a heterogeneous design methodology is obviously very important. Each model must in fact be capable of supporting the desired techniques, and must be detailed enough to provide answers to the specific questions under consideration for the particular subsystems it applies to. An equally important choice has to do with the way the levels of abstraction are connected, or, in other words, with the abstraction and refinement functions that are used to relate the models. In general, many forms of abstraction and refinement are possible. In practice, only those that preserve certain properties of interest are useful. In particular, we are interested in abstractions that preserve the refinement relationship $\preceq$ when moving from a more abstract model to a more concrete one. More formally, assume p and q are agents in a model $\mathcal{Q}$, and that p' and q' are the corresponding abstractions in a model $\mathcal{Q}'$. Then we say that the abstraction preserves the refinement relationship from the abstract to the concrete model if $p' \preceq q'$ implies that also $p \preceq q$.

This property is useful for several reasons. First, refinement verification can be used to establish that an agent satisfies some requirement by comparing it to a specification. It would therefore be at best inconvenient if the result of this verification were lost during a refinement step of the methodology. In the worst case, it could lead to incorrect designs.

A second advantage has to do with the efficiency of refinement verification. The process is in fact potentially more efficient at the abstract level because of the lesser amount of information included in the model. An abstraction that preserves the refinement relationship can thus be used to translate a complex verification problem at the concrete level to a simpler problem at the abstract level. This translation is conservative: while the loss of information may make it impossible to establish a refinement relationship between the abstracted agents, it ensures that when the relationship is indeed established it also holds at the concrete level. In other words, false positive results are ruled out.

We say that an abstraction is *proper* if knowing the abstraction of an agent p is not sufficient to determine p uniquely. A proper abstraction thus loses information when translating from one model to another. Proper abstractions are most useful in the context of refinement verification. Assume, in fact, that a model $\mathcal{Q}'$ is related to a model $\mathcal{Q}$ by a refinement preserving abstraction which is *not* proper. Then, assuming the complexity of computing the abstraction and its inverse is negligible compared to the complexity of checking refinement, verification in $\mathcal{Q}$ is as efficient as in $\mathcal{Q}'$, since it is possible to simply translate back and forth between the two models. Abstractions that are not proper are therefore not very interesting for refinement verification. In the following we will consider different classes of abstractions and discuss when they can be both refinement preserving and proper.

2.3 Monotonic functions

Abstraction functions that preserve the refinement relationship from the *concrete* to the *abstract* model are easy to find. Since the refinement relationship is an order, the function simply needs to be monotonic. For instance, if the concrete model is the set of real numbers $\mathbb{R}$, and the abstract model is the set of integers $\mathbb{Z}$, and the ordering $\preceq$ corresponds to the usual ordering, then the *ceiling* operator $\lceil r \rceil$ (the closest integer greater than or equal to r) preserves the ordering.

We are interested, however, in abstractions that preserve the ordering in the opposite direction. Monotonic functions from the abstract to the concrete model (i.e., refinement maps) have this property, but they cannot be used to abstract verification problems, since their inverse (the abstraction) will probably be a partial function. Instead, we need a function from the concrete to the abstract model that is intuitively "inverse" monotonic. The ceiling operator does not have this property. Assume, in fact, that $r = 2.8$ and $s = 2.3$. Then $\lceil r \rceil \preceq \lceil s \rceil$, however it is not the case that $r \preceq s$. In fact, no proper abstraction, based on a single function H, from a concrete model $\mathcal{Q}$ to a more abstract model $\mathcal{Q}'$ is refinement preserving, as shown by the following lemma.

Lemma 2.2. *Let $\mathcal{Q}$ and $\mathcal{Q}'$ be models of computation, and let $H : \mathcal{Q} \to \mathcal{Q}'$ be a function that preserves refinement from $\mathcal{Q}'$ to $\mathcal{Q}$. Then, $H(p_1) = H(p_2)$ implies that $p_1 = p_2$.*

The proof goes as follows. The assumption $H(p_1) = H(p_2)$ implies that $H(p_1) \preceq H(p_2)$ and that $H(p_2) \preceq H(p_1)$. Thus, since H preserves refinement from $\mathcal{Q}'$ to $\mathcal{Q}$, also $p_1 \preceq p_2$ and $p_2 \preceq p_1$. Therefore, $p_1 = p_2$.

Lemma 2.2 shows that if the function H preserves refinement from abstract to concrete, then H is injective, and therefore is not a proper abstraction since it has an inverse function. Thus, a function H is refinement preserving only if it implies no loss of information when moving between the two levels. Continuing our previous example, no function

$f : \mathbb{R} \to \mathbb{Z}$ preserves the ordering from $\mathbb{Z}$ to $\mathbb{R}$, since there exists no injection from $\mathbb{R}$ to $\mathbb{Z}$.

2.4 Galois connections

One way to address the problem is to consider abstractions whose structure is more complex than that of a simple function. One abstraction of this kind, which has been extensively studied in the literature, is a *Galois connection* [9]. A Galois connection between two models $\mathcal{Q}$ and $\mathcal{Q}'$ consists of an abstraction function α from $\mathcal{Q}$ to $\mathcal{Q}'$, and of a concretization function γ from $\mathcal{Q}'$ to $\mathcal{Q}$. A pair $\langle \alpha, \gamma \rangle$ forms a Galois connection whenever for all $p \in \mathcal{Q}$ and $p' \in \mathcal{Q}'$,

$$\alpha(p) \preceq p' \iff p \preceq \gamma(p').$$

The defining property of a Galois connection can be rephrased in the following terms.

Theorem 2.3 ([12], Proposition 4). *Let D and D' be partially ordered sets of agents. Then $\langle \alpha, \gamma \rangle$ is a Galois connection from D to D' if and only if*

- *α and γ are monotonic, and*
- *for all $p \in D$ and $p' \in D'$,*

$$p \preceq \gamma(\alpha(p))$$
$$\alpha(\gamma(p')) \preceq p'.$$

Theorem 2.3 establishes a relation that can be used to connect the order of the two models. However, the application of Galois connections to refinement verification is restricted by the following result.

Theorem 2.4. *Let $\langle \alpha, \gamma \rangle$ be a Galois connection between partially ordered sets of agents D and D', and let q be an agent of D. Then, $q = \gamma(\alpha(q))$ if and only if for all agents p, $\alpha(p) \preceq \alpha(q) \Rightarrow p \preceq q$.*

The proof goes as follows. Assume $q = \gamma(\alpha(q))$ and let p be an agent such that $\alpha(p) \preceq \alpha(q)$. Then, since γ is monotonic, $\gamma(\alpha(p)) \preceq \gamma(\alpha(q))$. By Theorem 2.3, $p \preceq \gamma(\alpha(q))$. By hypothesis, $p \preceq q$. For the reverse direction, assume that for all agents p, $\alpha(p) \preceq \alpha(q) \Rightarrow p \preceq q$. By Theorem 2.3, both $q \preceq \gamma(\alpha(q))$ and $\alpha(\gamma(\alpha(q))) \preceq \alpha(q)$. Therefore, by hypothesis, $\gamma(\alpha(q)) \preceq q$. Hence, $q = \gamma(\alpha(q))$.

Theorem 2.4 shows that a Galois connection preserves the refinement relation from abstract to concrete only if there is no loss of information when the abstraction is applied to the specification. To state it differently, the abstract model must be able to represent the specification exactly. In the other cases, the verification methodology is unsound and may lead to false positive results. Our example illustrates this. If we let $\alpha(r) = \lceil r \rceil$, and take the identity function $\gamma(r') = r'$ as the concretization function, then $\langle \alpha, \gamma \rangle$ forms a Galois connection from $\mathbb{R}$ to $\mathbb{Z}$. Thus, if r and s are real numbers, and if we know that $\alpha(r) \preceq \alpha(s)$, then, since γ is monotonic, $\gamma(\alpha(r)) \preceq \gamma(\alpha(s))$. By Theorem 2.3, we conclude that $r \preceq \gamma(\alpha(s))$. However, we may conclude that $r \preceq s$ only if $s = \gamma(\alpha(s))$, i.e., s is an integer.

2.5 Conservative approximations

Conservative approximations get around the problem outlined above by employing two abstraction functions, instead of just one. The first function, usually denoted Ψ_u, is applied to the implementation p, while the second function, denoted Ψ_l, is applied to the specification q. The pair (Ψ_l, Ψ_u) forms a conservative approximation whenever $\Psi_u(p) \preceq \Psi_l(q)$ implies $p \preceq q$. Thus, by definition, a conservative approximation always preserves the refinement relationship from the abstract to the concrete model. Unlike Galois connections, the two functions of a conservative approximation are both abstractions from the concrete model $\mathcal{Q}$ to the abstract model $\mathcal{Q}'$.

Conservative approximations enjoy several properties similar to those of Galois connections. For instance, if $\Psi' = (\Psi'_l, \Psi'_u)$ provides looser lower and upper bounds than a conservative approximation Ψ (i.e., if $\Psi'_l(p) \preceq \Psi_l(p)$ and $\Psi_u(p) \preceq \Psi'_u(p)$ for all p), then Ψ' is also a conservative approximation. Also, the functional composition of two conservative approximations yields another conservative approximation. Usually a conservative approximation $\Psi = (\Psi_l, \Psi_u)$ has the additional property that $\Psi_l(p) \preceq \Psi_u(p)$ for all p, but this is not required. Also, having Ψ_l and Ψ_u be monotonic (relative to the ordering on agents) is common, but not required.

Unlike abstractions based on a single function, conservative approximations need not be injective to preserve refinement. To continue our example of an abstraction from the set of real numbers $\mathbb{R}$ to the set of integers $\mathbb{Z}$, we define

$$\Psi_u(r) = \lceil r \rceil,$$
$$\Psi_l(s) = \lfloor s \rfloor,$$

where now $\lfloor s \rfloor$ is the *floor* of s (the closest integer *less than or equal to* s). Clearly, if $\lceil r \rceil \preceq \lfloor s \rfloor$, then $r \preceq s$ (which shows the pair forms a conservative approximation). However, neither $\Psi_u(x)$, nor $\Psi_l(x)$, nor the two in combination, are sufficient to determine x uniquely. Hence, the conservative approximation is indeed a proper abstraction. This can be accomplished because *different* abstraction functions are applied to the implementation and to the specification. In particular, the abstraction Ψ_u acts as an *upper bound* of the implementation, while the abstraction Ψ_l acts as a *lower bound* of the specification.

3 An example: Continuous and discrete time

To illustrate our definitions, we extend our example of continuous versus discrete numbers and build models of computation $\mathcal{Q}^C$ and $\mathcal{Q}^D$ that can be used to model event-based systems in continuous and discrete time.

In the continuous model, a pair (a, τ) is an event that denotes the occurrence of an action $a \in \mathcal{A}$, at a time $\tau \in \mathbb{R}$ (where $\mathcal{A}$ is some universe of actions). Each behavior of an agent consists of a set of events, such as

$$x = \{(a, 1.4), (b, 2.1), (c, 2.3), (b, 3.01), \ldots\}.$$

An agent p can thus be seen simply as a set P of behaviors of this sort. In addition, we characterize an agent by two disjoint sets $I \subseteq \mathcal{A}$ and $O \subseteq \mathcal{A}$ of input and output actions, which together form its alphabet A. The behaviors P of an agent $p = (I, O, P)$ are restricted

to events that perform actions in A. In order to complete the model, we must define operators such as projection and parallel composition. However, we defer these definitions to Section 4 when we discuss the compositionality properties of abstractions. The construction of an agent as a set of behaviors is quite general and can be employed with other models of behavior. In addition, the model may sometimes restrict the acceptable sets of behaviors for agents to express requirements such as receptiveness, or to avoid certain phenomena, such as Zeno behaviors. We will not, however, be concerned with these aspects in this paper.

It is straightforward to construct a similar, but more abstract, model of computation $\mathcal{Q}^D$ to be used for event-based systems in *discrete* time. It is in fact sufficient to restrict the time stamp τ to the set of integers $\mathbb{Z}$. This model is more abstract since any discrete time behavior is also a continuous time behavior, but not the other way around.

Our continuous and discrete time models can be seen as instances of the general definition of a Tagged-Signal Model (TSM) [17]. In TSM, a model of computation is constructed in a fixed way by considering a set of values V, and a set of tags T. The set of values represents the type of data that can be exchanged by objects in the model. The set of tags, on the other hand, carries an order relationship that is used in the model to encode the particular notion of time, or, more properly, of precedence. An *event* is represented by the pair $\langle t, v\rangle$, where $t \in T$ tags the "time" of the event, and $v \in V$ provides the new value. Processes are constructed by aggregating events into signals. In our examples, the set of tags T corresponds to $\mathbb{R}$ for the continuous time model, and to $\mathbb{Z}$ for the discrete time model. Here we present them in a more general form (our models are, in fact, agent algebras [24]) because we are interested in variations of these models that cannot be expressed in TSM. More importantly, our focus is on building relationships between these models. However, we are not aware of a general theory that explains the relation between different models expressed in TSM. The specialization of conservative approximations to this case is part of our future work.

Before we construct abstractions from the continuous time to the discrete time domain we must first define what it means in these models for an agent to refine another agent. Because agents in this case are simply sets of behaviors, specifications (which are themselves agents) can be seen as representing the set of *acceptable* behaviors. Any agent that contains only behaviors that are acceptable can be considered a *refinement* of the specification. Refinement is thus simply behavior containment. This is consistent with formal methods based on automata and language containment [7, 11]. More formally, an implementation $p = (I, O, P)$ refines a specification $q = (I, O, Q)$, written $p \preceq q$, whenever $P \subseteq Q$. This definition requires that p and q have the same set of inputs I and outputs O for the refinement to hold. A more elaborate model might consider relaxing this requirement, which is used here to simplify the exposition.

Abstractions from the continuous to the discrete time models may be constructed by first considering a function h that transforms a continuous time behavior into a discrete time behavior. One such abstraction simply truncates the real time stamp of the events in a continuous time behavior by applying the floor operator $\lfloor\tau\rfloor$ to obtain a discrete time tag, and therefore a discrete time behavior. For instance,

$$h(\{(a, 1.4), (b, 2.1), (c, 2.3), (b, 3.01), \ldots\}) = \{(a, 1), (b, 2), (c, 2), (b, 3), \ldots\}.$$

The abstraction $H(p)$ of an agent $p = (I, O, P)$ can then be obtained by extending h to sets of behaviors:

$$H(p) = (I, O, h(P)).$$

Here, clearly, $H(p)$ is a discrete time agent that is an abstract representation of p. In fact, the behaviors of p that have events that differ only for time stamps that are contained in the same integral interval are mapped onto the same behavior in $H(p)$. For example, the continuous time events $(a, 2.1)$ and $(a, 2.2)$ correspond to the *same* event $(a, 2)$ in the discrete time model. Consequently, the relative ordering of two actions a and b may be lost.

A Galois connection between $\mathcal{Q}^C$ and $\mathcal{Q}^D$ can be constructed as follows. We select $\alpha = H$ as the abstraction function. For the concretization of an agent $p' = (I, O, P')$ we use the inverse image of h extended to sets, and thus we define

$$\gamma(p') = (I, O, \{x : h(x) \in P'\}) = (I, O, h^{-1}(P')),$$

where x ranges on the set of continuous time behaviors ($2^{A\times\mathbb{R}}$). In other words, the concretization p of p' contains all and only the behaviors whose abstraction is in P'. It follows from the definitions that α and γ form a Galois connection between $\mathcal{Q}^C$ and $\mathcal{Q}^D$ relative to the behavior containment order of agents. In fact, both α and γ are monotonic, and the set theoretic properties

$$P \subseteq h^{-1}(h(P)),$$
$$P' = h(h^{-1}(P')),$$

can be used to prove that the conditions of Theorem 2.3 are satisfied.

Similarly, we construct a conservative approximation between the continuous and the discrete time model. Recall that the order for the agents is expressed in terms of set containment, and is structurally very different from the traditional order on $\mathbb{R}$ and $\mathbb{Z}$ that we employed in our earlier example. Thus, using a combination of the floor and the ceiling operator will not work. In fact, the function $H(p)$ that employs the floor operator can be shown to be an upper bound since, in this case, every behavior of p is represented by at least one abstract behavior of $H(p)$. In turn, $H(p)$ represents potentially *more* behaviors than p, corresponding to all its possible concretizations. Thus, we choose

$$\Psi_u^C(p) = H(p) = (I, O, h(P)),$$

which corresponds to α above. Conversely, for a lower bound, we would like the discrete agent to include abstractions of *only* the behaviors that are contained in the continuous time agent. Thus, a behavior x' is included in the abstraction $\Psi_l^C(p)$ only if p contains *all* the concretizations of x'. This can be represented formally as

$$\Psi_l^C(p) = (I, O, h(P) - h(\mathcal{B}(A) - P)),$$

where $\mathcal{B}(A)$ is the set of all the behaviors that can be constructed with actions in the set A (the alphabet of the agent). For example, the behavior $\{(a, 1)\}$ is in $\Psi_l^C(p)$ only if p contains a behavior $\{(a, \tau)\}$ for every $\tau \in [1, 2)$. Note that Ψ_l does not correspond to either α or γ above.

Defining abstraction functions that satisfy the properties for Galois connections and conservative approximations (Sections 2.4 and 2.5) may appear to be difficult. Here, we have used a process to derive Galois connections and conservative approximations based on extending a function on behaviors to act on sets of behaviors. This process can be generalized by using the notion of an axiality [24], which is similar to the pre and

post images of a binary relation proposed by Loiseaux et al. [20], and to the optimistic and pessimistic process abstractions of Negulescu [22]. This construction, which guarantees that the resulting abstraction have the required properties, can be applied to most behavior-based models. Because reasoning on individual behaviors is easier than acting directly on agents, this construction greatly simplifies the process of creating abstractions between two agent models [5, 6, 24]. Conservative approximations, however, are not just limited to axialities [24], and can be derived using other set theoretic constructions [3].

Functions other than h (which is based on the floor operator) can be used to relate our continuous time and discrete time model. For example, we may consider *rounding* rather than *truncating* the real time stamps. Different functions give rise to different Galois connections and conservative approximations. These, in turn, can be used to represent different implementation strategies in a design flow. While the techniques described in this paper are independent of the particular form of the functions, this discussion is beyond the scope of this paper.

We can use our continuous and discrete time models to illustrate the differences between conservative approximations and Galois connections. Consider, for example, the continuous time specification that says that an action b must always be preceded by a corresponding action a. This specification can be expressed in $\mathcal{Q}^C$ by an agent q that contains all and only the behaviors that have such property. Ignoring the presence of actions other than a and b, one such behavior is, for example,

$$x = \{(a, 1.1), (b, 2.3), (a, 3.4), (b, 3.8), \ldots\}.$$

The Galois connection makes use of the abstraction function α (which corresponds to the upper bound Ψ_u^C of the conservative approximation). The behavior x above, for example, is represented in $\alpha(q)$ as

$$x' = \{(a, 1), (b, 2), (a, 3), (b, 3), \ldots\}.$$

The order between $(a, 3.4)$ and $(b, 3.8)$ has been lost. As a result, the concretization $\gamma(\alpha(q))$ may contain behaviors such as

$$y = \{(a, 1.1), (b, 2.3), (b, 3.8), (a, 3.9), \ldots\},$$

which clearly violate our specification. In other words, even if one shows that $\alpha(p) \preceq \alpha(q)$, it may still be the case that $p \not\preceq q$.

The above verification technique based on Galois connections is unsound because q, our specification, is not represented exactly in $\mathcal{Q}^D$. Conservative approximations avoid this problem by explicitly using the lower bound. Because y above does not satisfy the specification, then clearly y is not a behavior of q. Thus, since x' is an abstraction of y, it follows that x' is not included in the lower bound $\Psi_l^C(q)$ of the specification, or else q would have to include y. Therefore, if y were a behavior of some implementation p, then x' would be a behavior of $\Psi_u^C(p)$ (since we are using the upper bound for the implementation), and $\Psi_u^C(p) \preceq \Psi_l^C(q)$ would not hold. The same occurs if x, rather than y, is a behavior of p, even though in this case p would satisfy the specification. Hence, the conservative approximation correctly concludes that the abstract model is unable to tell whether p is a refinement of q or not.

The lower bound $\Psi_l^C(q)$ of the specification is, however, not empty. It contains, for instance, all those behaviors in which the action a and the corresponding action b occur in

two different time intervals $[t, t+1)$, where $t \in \mathbb{Z}$, and are ordered according to the specification. For such behaviors, the concretization cannot possibly invert the order relationship between the actions. Hence, if $\Psi_u^C(p) \preceq \Psi_l^C(q)$, then necessarily $p \preceq q$. In other words, if the implementation p is "slow enough" compared to the abstract model, verification at the abstract level is possible. Unlike the abstract interpretation, the conservative approximation automatically detects this condition.

Our continuous time specification could be represented exactly at the abstract level if, for example, we were to use *sequences* of events as behaviors, as opposed to *sets* of events. This amounts to decreasing the level of abstraction of the discrete time model. In that case, in fact, the order of the actions can be preserved, and the verification problem can be addressed using Galois connections. This technique, however, becomes again unsound if we were to consider a different specification in continuous time, such as one that requires a certain real-time deadline, or a certain response time, to be met by the implementation (Burch discusses several models for both continuous and discrete time [3]). For this case, Galois connections may again lead to false positive verification results. The situation can be fixed by yet again lowering the level of abstraction of the discrete time model. This dependency between the verification methodology and the level of abstraction of the models employed is, however, troublesome. In particular, it is contrary to the principle of orthogonalization of concerns, whereby we would like to choose our models, the specification and verification techniques independently, while ensuring correct results. In addition, the models employed in a design are often fixed and determined by the particular tools used in the design flow. Conservative approximations, on the other hand, do not suffer from this problem. Specifically, a conservative approximation guarantees that if a verification problem can be positively solved at the abstract level, then it holds at the concrete level, as well. This fact allows the verification methodology to adapt to the specific models being used, while guaranteeing correct results in the cases that can be handled at the abstract level. We therefore view conservative approximations and abstract interpretations as related, but complementary, concepts.

4 Compositionality properties

So far, our discussion has revolved around the relationships that exist between individual agents. Systems are, however, typically constructed by composing agents, using the operators of projection, renaming and parallel composition introduced in Section 2.1. In this section we introduce abstractions that "behave well" with respect to these operators. Before we do so, we complete the description of the continuous and discrete time models by defining the exact form of these operators. Other definitions are possible, these are specific choices that we make for these models.

Consider the continuous time model of computation $\mathcal{Q}^C$. Projection may be defined by simply removing from the behaviors of an agent those events that correspond to actions that must not be retained. For example, if $\{a, c\}$ is the set of actions that we want to retain, then the projection of the behavior $x = \{(a, 1.4), (b, 2.1), (c, 2.3), (b, 3.01), \ldots\}$ is

$$proj(\{a, c\})(x) = \{(a, 1.4), (c, 2.3), \ldots\}.$$

Renaming is defined similarly by extending to events and to the sets I and O the application of a renaming function r, i.e., a bijection on the set of signals $\mathcal{A}$. For instance, if a renaming

function r maps signal a to in_1, signal b to in_2 and signal c to out, then

$$rename(r)(x) = \{(in_1, 1.4), (in_2, 2.1), (out, 2.3), (in_2, 3.01), \ldots\}.$$

Parallel composition is more complex. Let $p_1 = (I_1, O_1, P_1)$ and $p_2 = (I_2, O_2, P_2)$ be agents with alphabets $A_1 = I_1 \cup O_1$ and $A_2 = I_2 \cup O_2$, respectively. The parallel composition $p = p_1 \parallel p_2$ is defined only if the sets O_1 and O_2 are disjoint, to ensure that only one agent drives each action. When defined, an action is an output of the parallel composition if it is an output of either agent. Conversely, it is an input if it is an input of either p_1 or p_2, and it is not an output of the other agent. Thus

$$\begin{aligned} O &= O_1 \cup O_2, \\ I &= (I_1 \cup I_2) - (O_1 \cup O_2). \end{aligned}$$

The alphabet of the composition is therefore $A = A_1 \cup A_2$. A behavior y is part of the parallel composition if and only if the projection of y to the alphabet of p_1 is a behavior of p_1, and the projection to the alphabet of p_2 is a behavior of p_2. Formally,

$$P = \{y \in \mathcal{B}(A) : proj(A_1)(y) \in P_1 \wedge proj(A_2)(y) \in P_2\},$$

where $\mathcal{B}(A)$ denotes the set of behaviors that have events with actions in A. This definition ensures that the behaviors of the composition are consistent with the behaviors of each component. The same result can be obtained by taking the intersection of the sets of behaviors of the individual agents, after an operation of inverse projection to the common alphabet. This is, for example, the way parallel composition is defined in the Tagged-Signal Model [17].

The operators for the discrete time model $\mathcal{Q}^D$ are defined similarly. In fact, these definitions can be equally applied to all models of computation based on the notion of behavior or execution. As already pointed out, however, the results presented in this paper are more general and apply also to models whose agents are not derived as sets of executions or behaviors [24].

Different operators can be used in sequence to build complex systems. We handle this case in the standard way, by constructing expressions that denote the order of application of the operators. For instance, a system may be constructed by first instantiating two agents p_1 and p_2 through a renaming operation, then taking their parallel composition, and finally by hiding certain internal signals (or actions) that are not relevant to the environment. This can be represented by an expression E of the form

$$E = proj(A)(rename(r_1)(p_1) \parallel rename(r_2)(p_2)),$$

where A, r_1 and r_2 are appropriate projection sets and renaming functions. The expression E can be evaluated by applying the operators in their order to yield an agent in the model, which stands for the entire system. In the rest of the paper, we will use the symbol E to denote both the expression and its evaluation. The context will make clear what is intended.

In general we say that an abstraction H from $\mathcal{Q}$ to $\mathcal{Q}'$ is *compositional* if it commutes with the operators of the models. The terminology derives from the application of the composition operator. In this case, we require that for all agents p_1 and p_2,

$$H(p_1 \parallel p_2) = H(p_1) \parallel H(p_2),$$

where the left hand side is evaluated with the composition operator of $\mathcal{Q}$, while the right hand side with the one of $\mathcal{Q}'$. The condition above amounts to restricting our attention to only homomorphisms between models of computation. In practice, however, weaker conditions are sufficient to guarantee that certain results of interest hold. The next two sections show how to extend Galois connections and conservative approximations to deal with compositionality. In the following, we also rely on the assumption that the operators are monotonic relative to the refinement order of the model. This is a common assumption when dealing with compositionality, and one that is satisfied by our continuous and discrete time model.

4.1 Abstract interpretations

The extension of Galois connections to compositional functions is traditionally called an *abstract interpretation*. Abstract interpretations were originally developed for static analysis of sequential programs in optimizing compilers [8]. They have also been used for abstracting and formally verifying models of both sequential and reactive systems. An abstract interpretation between models of computation is essentially a Galois connection with certain additional properties. It can be defined as follows.

Definition 4.1 (Abstract interpretation). Let $\mathcal{Q}$ and $\mathcal{Q}'$ be models of computation. Then $\mathcal{Q}'$ is an abstract interpretation of $\mathcal{Q}$ if and only if there exists a Galois connection $\langle \alpha, \gamma \rangle$ from $\mathcal{Q}$ to $\mathcal{Q}'$ such that for all agents p', p'_1 and p'_2 in $\mathcal{Q}'$,

1. if $p'_1 \parallel p'_2$ is defined, then $\alpha(\gamma(p'_1) \parallel \gamma(p'_2)) \preceq p'_1 \parallel p'_2$,
2. if $proj(B)(p')$ is defined, then $\alpha(proj(B)(\gamma(p'))) \preceq proj(B)(p')$,
3. if $rename(r)(p')$ is defined, then $\alpha(rename(r)(\gamma(p'))) \preceq rename(r)(p')$.

The abstraction and concretization functions α and γ from $\mathcal{Q}^C$ to $\mathcal{Q}^D$ introduced in Section 3 satisfy the three properties of Definition 4.1. Thus, $\mathcal{Q}^D$ is an abstract interpretation of $\mathcal{Q}^C$.

Abstract interpretations are used in program analysis because they preserve the application of the operators from the abstract model to the concrete model. In fact, assume that $\mathcal{Q}'$ is an abstract interpretation of $\mathcal{Q}$ by a Galois connection $\langle \alpha, \gamma \rangle$, and that E is an expression. We use the notation $E[p/\alpha(p)]$ to denote the expression E' in the abstract domain that is structurally identical to E (it uses the same operators and in the same order), but where each leaf agent p in E has been replaced by its abstraction $\alpha(p)$. It can be shown that if $E[p/\alpha(p)]$ is defined, then

$$E \preceq \gamma(E[p/\alpha(p)]).$$

Hence, abstract interpretations can be used to approximate the evaluation of an expression at the concrete level by the concretization of the evaluation of the corresponding expression at the abstract level. The abstract interpretation guarantees that the result computed at the concrete level conforms to (or refines) the one computed at the abstract level. This is useful when the evaluation of E is easy (or at least feasible) in $\mathcal{Q}'$, while it is difficult (or impossible) in $\mathcal{Q}$. If the concretization of the abstract evaluation satisfies the specification, then so does the evaluation of the original expression (by transitivity).

Since abstract interpretations are derived from Galois connections, they again fail to preserve refinement verification results. This can be accomplished with the compositional version of conservative approximations.

4.2 Compositional conservative approximations

A refinement verification problem is often of the form $E \preceq q$, where q is the specification and E is an expression using the operators of the model. Computing $\Psi_u(E)$ involves evaluating the expression E in the concrete domain, a potentially expensive operation. A compositional conservative approximation allows us to avoid this computation by translating the expression into the abstract domain. As an example, consider the verification problem

$$proj(A)(p_1 \parallel p_2) \preceq q,$$

where p_1, p_2 and q are agents in $\mathcal{Q}$. This corresponds to checking whether an implementation consisting of two components p_1 and p_2 (along with some internal signals that are removed by the projection operation) satisfies the specification q. To take advantage of a conservative approximation Ψ, one would need to check whether

$$\Psi_u(proj(A)(p_1 \parallel p_2)) \preceq \Psi_l(q),$$

which requires the evaluation of the composition $p_1 \parallel p_2$ (in fact of the whole expression) in the concrete domain before the abstraction is applied. We say that a conservative approximation Ψ is a *compositional* conservative approximation if showing

$$proj(A)(\Psi_u(p_1) \parallel \Psi_u(p_2)) \preceq \Psi_l(q)$$

is sufficient to show that the original implementation satisfies its specification. Here, the abstraction is applied to the components first, and the expression (including the composition) is then evaluated in the abstract model, where the computation may be simpler. The following definition makes this notion precise.

Definition 4.2 (Compositional conservative approximations). A conservative approximation $\Psi = (\Psi_l, \Psi_u)$ from $\mathcal{Q}$ to $\mathcal{Q}'$ is a *compositional conservative approximation* if and only if for all expressions E and for all agents $q \in \mathcal{Q}$,

$$E[p/\Psi_u(p)] \preceq \Psi_l(q) \Rightarrow E \preceq q,$$

where the notation $E[p/\Psi_u(p)]$ again indicates that every agent p in E must be replaced by the corresponding agent $\Psi_u(p)$.

Again, a compositional conservative approximation is defined so that it has the required properties. The following theorem provides weaker sufficient conditions (which are easier to verify) that the upper bound of a conservative approximation must satisfy in order for the approximation to also be compositional.

Theorem 4.3. *Let $\Psi = (\Psi_l, \Psi_u)$ be a conservative approximation from $\mathcal{Q}$ to $\mathcal{Q}'$. If the following propositions S1 through S3 are satisfied for all agents p, p_1 and p_2 in $\mathcal{Q}$, then Ψ is a compositional conservative approximation.*

S1. If $\Psi_u(p_1) \parallel \Psi_u(p_2)$ is defined, then $\Psi_u(p_1 \parallel p_2) \preceq \Psi_u(p_1) \parallel \Psi_u(p_2)$.
S2. If $proj(B)(\Psi_u(p))$ is defined, then $\Psi_u(proj(B)(p)) \preceq proj(B)(\Psi_u(p))$.
S3. If $rename(r)(\Psi_u(p))$ is defined, then $\Psi_u(rename(r)(p)) \preceq rename(r)(\Psi_u(p))$.

The conservative approximation Ψ^C from $\mathcal{Q}^C$ to $\mathcal{Q}^D$ is compositional. Recall, in fact, that the upper and lower bound of Ψ^C are derived from a function h that truncates the time stamp of all events in a behavior by applying the floor operator $\lfloor \tau \rfloor$. We have shown in Section 3 that, in order to obtain a conservative approximation, it is sufficient to extended the function h, which is a homomorphism on behaviors, to *sets* of behaviors. There is a general result that guarantees, by applying Theorem 4.3, that this construction produces a compositional conservative approximation [3, 24]. This is true regardless of the behavior model, as long as it satisfies certain basic properties related to the requirements of our agent models. In particular, and to illustrate our argument, if we interpret the parallel composition operator as intersection of sets of behaviors, and refinement as set containment, proposition S1 above follows from the property

$$h(P_1 \cap P_2) \subseteq h(P_1) \cap h(P_2),$$

that holds in general for any function h extended to sets. This result dramatically simplifies the problem of constructing compositional conservative approximations between agent models.

In a refinement verification problem we might sometimes be interested in applying compositionality to the specification as well as to the implementation side of the inequality. In this case we talk about a *fully* compositional conservative approximation.

Definition 4.4 (Fully compositional conservative approximations). A conservative approximation $\Psi = (\Psi_l, \Psi_u)$ from $\mathcal{Q}$ to $\mathcal{Q}'$ is a *fully compositional conservative approximation* if and only if for all expressions E_1 and E_2,

$$E_1[p/\Psi_u(p)] \preceq E_2[p/\Psi_l(p)] \Rightarrow E_1 \preceq E_2.$$

A result similar to Theorem 4.3 can be derived by adding appropriate conditions to the lower bound of a conservative approximation.

Theorem 4.5. *Let $\Psi = (\Psi_l, \Psi_u)$ be a conservative approximation from $\mathcal{Q}$ to $\mathcal{Q}'$ satisfying propositions S1 through S3 and S4 through S6 below. Then Ψ is a fully compositional conservative approximation.*

S4. If $\Psi_l(p_1 \parallel p_2)$ is defined, then $\Psi_l(p_1) \parallel \Psi_l(p_2) \preceq \Psi_l(p_1 \parallel p_2)$.
S5. If $\Psi_l(proj(B)(p))$ is defined, $proj(B)(\Psi_l(p)) \preceq \Psi_l(proj(B)(p))$.
S6. If $\Psi_l(rename(r)(p))$ is defined, then $rename(r)(\Psi_l(p)) \preceq \Psi_l(rename(r)(p))$.

Again, it can be shown that the conservative approximation Ψ^C from $\mathcal{Q}^C$ to $\mathcal{Q}^D$ satisfies S4 through S6, and therefore is fully compositional. However, unlike propositions S1 through S3, the corresponding properties S4 through S6 are not necessarily satisfied by a conservative approximation derived from a homomorphism on behaviors. This may occur since the projection at the abstract level may overestimate the projection at the concrete level (because

it has less information), instead of underestimating it as required. This fact affects the properties of the refinement maps, as discussed below.

5 Inverse approximations

As we have discussed, if $\Psi = (\Psi_l, \Psi_u)$ is a conservative approximation from $\mathcal{Q}$ to $\mathcal{Q}'$, then $p' = \Psi_u(p)$ represents a kind of upper bound on p. It is instructive to investigate whether there is an agent in $\mathcal{Q}$ that is represented exactly by p' rather than just being bounded by p'. If no agent in $\mathcal{Q}$ can be represented exactly, then Ψ is abstracting away too much information to be of much use for verification. If every agent in $\mathcal{Q}$ can be represented exactly, then Ψ_l and Ψ_u are equal and are isomorphisms from $\mathcal{Q}$ to $\mathcal{Q}'$. These extreme cases illustrate that the amount of abstraction in Ψ is related to what agents p are represented exactly by $\Psi_u(p)$ and $\Psi_l(p)$.

To formalize what it means to be represented exactly in the context of conservative approximations, we define the inverse Ψ_{inv} of the conservative approximation Ψ. The inverse of an approximation is a function from the abstract model $\mathcal{Q}'$ to the concrete model $\mathcal{Q}$ that, as we shall see in this section, completes the relationships between $\mathcal{Q}$ and $\mathcal{Q}'$ by establishing a refinement map across the models. The combination of abstraction and refinement will be used later in Section 7 to discuss the polymorphic properties of agents employed in heterogeneous systems.

Normal notions of the inverse of a function are not adequate for constructing the inverse of a conservative approximation Ψ, since Ψ is a pair of functions. Our notion of an inverse is thus based on the following result.

Lemma 5.1. *Let $\mathcal{Q}$ and $\mathcal{Q}'$ be models of computation, and let (Ψ_l, Ψ_u) be a conservative approximation from $\mathcal{Q}$ to $\mathcal{Q}'$. For all p_1 and p_2 in $\mathcal{Q}$, if $\Psi_l(p_1) = \Psi_u(p_1) = p'$ and $\Psi_l(p_2) = \Psi_u(p_2) = p'$, then $p_1 = p_2$.*

Lemma 5.1 shows that when the upper and the lower bound coincide for a particular agent p, then, intuitively, the abstraction p' is an exact representation of p. To put it another way, p does not use any of the additional information provided by the concrete level, since it can be determined uniquely from its abstraction p'. It is therefore natural to define $\Psi_{inv}(p') = p$, where p is the agent in $\mathcal{Q}$ such that $\Psi_u(p) = \Psi_l(p) = p'$. If $\Psi_l(p) \neq \Psi_u(p)$, then p is not represented exactly in $\mathcal{Q}'$. In this case, p is not in the image of Ψ_{inv}.

Definition 5.2 (Inverse of a conservative approximation). Let $\Psi = (\Psi_l, \Psi_u)$ be a conservative approximation from $\mathcal{Q}$ to $\mathcal{Q}'$. For $p' \in \mathcal{Q}'.D$, the inverse $\Psi_{inv}(p')$ is defined and is equal to p if and only if $\Psi_l(p) = \Psi_u(p) = p'$.

It follows from the definition that, when Ψ_{inv} is defined, the following identity holds:

$$\Psi_l(\Psi_{inv}(p')) = \Psi_u(\Psi_{inv}(p')).$$

The function Ψ_{inv} need not be defined for all p'. This may happen, for example, if the model $\mathcal{Q}'$ includes information that cannot be expressed exactly in $\mathcal{Q}$. In that case, Ψ_{inv} is a partial function and is only defined for the agents that have an exact representation in both models. When an agent has an exact representation in $\mathcal{Q}$ and $\mathcal{Q}'$, we say that it can be used *indifferently* in the two models, or that it is *polymorphic*. This is because the agent makes no

assumption regarding its behavior based on information that can be expressed exclusively in either model of computation. However, the representation of the agent in $\mathcal{Q}$ and $\mathcal{Q}'$ is, in general, different. Thus, this notion extends our ability to reuse agents across models that employ different representations. We will discuss this aspect of our theory in more details in Section 7.

A conservative approximation thus induces its own inverse in the form of a (possibly partial) refinement map. The inverse is uniquely determined, and, because of the defining properties of a conservative approximation, Ψ_{inv} is one-to-one, and, when restricted to the image of Ψ_{inv}, the functions Ψ_l and Ψ_u are equal and are the inverse of Ψ_{inv}. In addition, when defined, Ψ_{inv} is always monotonic and, if either Ψ_l or Ψ_u is also monotonic, it preserves the ordering of the agents in both directions. Hence, the inverse embeds the abstract model of computation (or at least the part of it where it is defined) into the more concrete model, in a way that is consistent with the chosen abstractions. Different conservative approximations between the same models may therefore induce different embeddings. This is again an indication of the importance of choosing the right abstraction for the problem at hand. The nature of the embedding, in fact, determines how one model is interpreted in terms of the other, and quantifies the amount of information lost during the abstraction.

As noted, the inverse of a conservative approximation enjoys several properties. The inverse can also be used to formally understand the roles of the upper and the lower bounds. Let p be an agent in a model $\mathcal{Q}$ such that $\Psi_{inv}(\Psi_l(p))$ and $\Psi_{inv}(\Psi_u(p))$ are defined (since p is arbitrary, $\Psi_l(p)$ and $\Psi_u(p)$ are not necessarily equal). From the defining properties of conservative approximations, it follows that

$$\Psi_{inv}(\Psi_l(p)) \preceq p \preceq \Psi_{inv}(\Psi_u(p)).$$

This fact makes precise the intuition that $\Psi_l(p)$ and $\Psi_u(p)$ represent a lower and an upper bound of p, respectively.

Every agent in the abstract model $\mathcal{Q}'$ corresponds, through each abstraction function, to a (possibly empty) set of agents in the concrete model $\mathcal{Q}$. These sets, in fact, form a partition of $\mathcal{Q}$, and they are therefore equivalence classes. For conservative approximations, because there are two abstraction functions, each agent $p' \in \mathcal{Q}'$ determines two equivalence classes in $\mathcal{Q}$: the class of the agents p such that $\Psi_u(p) = p'$, and the class of the agents p such that $\Psi_l(p) = p'$. It is interesting to examine the relationship between these equivalence classes and the inverse. It can be shown that the inverse $\Psi_{inv}(p')$ of p' is defined if and only the greatest element of the class induced by Ψ_u is equal to the least element the class induced by Ψ_l, and that, when defined, the inverse is equal to both. Formally, the following results holds.

Theorem 5.3. *Let $\Psi = (\Psi_l, \Psi_u)$ be a conservative approximation from $\mathcal{Q}$ to $\mathcal{Q}'$. Then the following two statements are equivalent:*

- *$\Psi_{inv}(p')$ is defined and is equal to p,*
- *$\max\{p_1 : \Psi_u(p_1) = p'\}$ and $\min\{p_1 : \Psi_l(p_1) = p'\}$ exist and are equal to p.*

This is an alternative characterization of the inverse of a conservative approximation which is useful to explain the role of the abstraction functions.

Using the above characterization we can determine the inverse Ψ^C_{inv} of the conservative approximation $\Psi^C = (\Psi^C_l, \Psi^C_u)$ from our continuous time model $\mathcal{Q}^C$ to the discrete time model $\mathcal{Q}^D$. Given $p' \in \mathcal{Q}^D$, the equivalence class induced by Ψ^C_u includes all agents $p \in \mathcal{Q}^C$

that have at least one concrete behavior for every abstract behavior of p', and at the same time none of the behaviors which are not concretizations of behaviors of p'. This set has a maximum element, which is the agent $\hat{p}$ that includes *all* the concretizations of the behaviors of p' (and nothing else). In addition, of those agents that include all the concretizations, but possibly more, $\hat{p}$ is the least, since it does not include any other behavior. Thus $\hat{p}$ is also the minimum of the equivalence class induced by Ψ_l^C. Hence, $\Psi_{inv}^C(p') = \hat{p}$. By these definitions, the inverse of the conservative approximation is equal to the concretization function of the Galois connection described in Section 3 (this is not accidental, see Section 6 below). Also, in this case, the inverse is always defined.

As an example, consider the agent p' that has only one behavior $\{(a, \tau')\}$, where $\tau' \in \mathbb{Z}$. Then its inverse is the agent $\hat{p}$ with the following set of behaviors:

$$P = \{\{(a, \tau)\} : \tau' \leq \tau < \tau' + 1\}.$$

Thus, as discussed, $\hat{p}$ is represented exactly in $\mathcal{Q}^D$ by p'. This is because $\hat{p}$ may non-deterministically execute the action a at any time in the interval $[\tau', \tau' + 1)$ (since it includes all such behaviors). The abstraction functions determine this interpretation, which is made explicit by their inverse. This also explains why the specification q in Section 3 could not be represented exactly in $\mathcal{Q}^D$. In fact, x' is not contained in $\Psi_l^C(q)$. If it were, q would have to contain behaviors, such as y in Section 3, where action b *precedes* action a in the time span from 3 to 4, which contradicts the specification. Hence $\Psi_u^C(q) \neq \Psi_l^C(q)$.

There exist several injective functions from $\mathcal{Q}^D$ to $\mathcal{Q}^C$. One could for instance define a different refinement map G where $G(p')$ above is the agent with only the behavior $\{(a, \tau)\}$, where $\tau = \tau'$ is the original integer time stamp interpreted as a real. This choice, however, is arbitrary relative to the abstraction functions. An equally valid choice would be to assign $\tau = \tau' + \epsilon$, where $0 \leq \epsilon < 1$. These functions are therefore *not* inverses of our conservative approximations. Nonetheless, G could potentially be the inverse of a different conservative approximation, provided that the abstraction functions and, potentially, the order on the agents are changed appropriately to reflect the intended implementation strategy. In this case, also the interpretation of what is represented exactly by the two models would change.

It is interesting to consider the compositionality properties of the inverse of a conservative approximation with respect to the operators of the models, as discussed for the abstraction functions in Section 4. We have already pointed out that the inverse of a conservative approximation is monotonic, and in fact, if the abstraction functions are also monotonic, it preserves the refinement relationship in both directions. In addition, when the upper bound of a conservative approximation satisfies S1 through S3 of Theorem 4.3, then we can prove similar properties for the inverse of the conservative approximation.

Theorem 5.4. *Let $\Psi = (\Psi_l, \Psi_u)$ be a compositional conservative approximation from $\mathcal{Q}$ to $\mathcal{Q}'$ satisfying S1 through S3. Let p'_1 and p'_2 be agents in $\mathcal{Q}'$ such that $\Psi_{inv}(p'_1)$ and $\Psi_{inv}(p'_2)$ are both defined. Then*

1. *If $\Psi_{inv}(p'_1 \parallel p'_2)$ is defined, then $\Psi_{inv}(p'_1) \parallel \Psi_{inv}(p'_2) \preceq \Psi_{inv}(p'_1 \parallel p'_2)$.*
2. *If $\Psi_{inv}(proj(B)(p'_1))$ is defined, then $proj(B)(\Psi_{inv}(p'_1)) \preceq \Psi_{inv}(proj(B)(p'_1))$.*
3. *If $\Psi_{inv}(rename(r)(p'_1))$ is defined, then $rename(r)(\Psi_{inv}(p'_1)) \preceq \Psi_{inv}(rename(r)(p'_1))$.*

Theorem 5.4 shows that the inverse of a compositional conservative approximation can be used as a concretization function, similarly to the one employed by an abstract interpretation,

to give an upper approximation of the evaluation of an expression (this fact will be made precise in Section 6).

Since the inverse Ψ_{inv} of a conservative approximation is one-to-one, it is natural to ask whether it also commutes with the operators, instead of just approximating them. This is the case for a particular class of fully compositional conservative approximations.

Theorem 5.5. *Let $\Psi = (\Psi_l, \Psi_u)$ be a fully compositional conservative approximation from $\mathcal{Q}$ to $\mathcal{Q}'$ satisfying propositions S1 through S3 and S4 through S6. Further assume that for all agents $p \in \mathcal{Q}$, $\Psi_l(p) \preceq \Psi_u(p)$. Then, for the subset of $\mathcal{Q}'$ where Ψ_{inv} is defined,*

$$\Psi_{inv}(p'_1 \parallel p'_2) = \Psi_{inv}(p'_1) \parallel \Psi_{inv}(p'_2),$$
$$\Psi_{inv}(rename(r)(p')) = rename(r)(\Psi_{inv}(p')),$$
$$\Psi_{inv}(proj(B)(p')) = proj(B)(\Psi_{inv}(p')).$$

In this case, the inverse is effectively a full embedding of the abstract model (or the portion of the abstract model where the inverse is defined) into the concrete model. In other words, the abstract model is *isomorphic* to a subset of the concrete model. We will consider this fact in more details when discussing polymorphism in Section 7.

6 Conservative approximations induced by Galois connections

In Section 1 we have argued that there exists a close relationship between conservative approximations, Galois connections and abstract interpretations. In this section we explore this relationship in more detail.

Let $\mathcal{Q}$ and $\mathcal{Q}'$ be two models of computation. Recall from Section 2 that a Galois connection from $\mathcal{Q}$ to $\mathcal{Q}'$ is composed of an abstraction function α from $\mathcal{Q}$ to $\mathcal{Q}'$ and of a concretization function γ from $\mathcal{Q}'$ to $\mathcal{Q}$. In contrast, a conservative approximation consists of a pair of functions Ψ_l and Ψ_u that go from $\mathcal{Q}$ to $\mathcal{Q}'$, with an induced inverse function Ψ_{inv} from $\mathcal{Q}'$ to $\mathcal{Q}$. We have discussed how the two functions Ψ_l and Ψ_u have a distinguished role as the abstraction of the specification and the implementation, respectively.

The relationship between Galois connections and conservative approximations can be understood by introducing a second Galois connection that takes the role of the lower bound of a conservative approximation. To function as a lower bound, the order properties of the Galois connection must be reversed. Such an "inverted" Galois connection is equivalent to a regular one that goes in the reverse direction, i.e., from $\mathcal{Q}'$ to $\mathcal{Q}$. In the rest of this paper, we will take advantage of this correspondence. For our notation, we will use symbols $\langle \alpha_u, \gamma_u \rangle$ for a Galois connection from $\mathcal{Q}$ to $\mathcal{Q}'$, and $\langle \gamma_l, \alpha_l \rangle$ for a Galois connection from $\mathcal{Q}'$ to $\mathcal{Q}$. In the case of the Galois connection from $\mathcal{Q}'$ to $\mathcal{Q}$, we use the symbol γ_l for the abstraction map, and α_l for the concretization map. This choice is made clear by our results on the correspondence between conservative approximations and abstract interpretations.

Our first result exactly characterizes the conditions under which a pair of Galois connections forms a conservative approximation.

Theorem 6.1. *Let $\langle \alpha_u, \gamma_u \rangle$ be a Galois connections from $\mathcal{Q}$ to $\mathcal{Q}'$ and $\langle \gamma_l, \alpha_l \rangle$ a Galois connection from $\mathcal{Q}'$ to $\mathcal{Q}$. Then, (α_l, α_u) is a conservative approximation from $\mathcal{Q}$ to $\mathcal{Q}'$ if and only if for all agents $p' \in \mathcal{Q}'$, $\gamma_u(p') \preceq \gamma_l(p')$.*

In that case, we say that (α_l, α_u) is the *conservative approximation induced by the pair of Galois connections* $\langle \alpha_u, \gamma_u \rangle$ and $\langle \gamma_l, \alpha_l \rangle$. The degree by which γ_u and γ_l differ influences the accuracy of the conservative approximation. In particular, the closer γ_l is to γ_u, the tighter the resulting conservative approximation, as shown by the next result.

Corollary 6.2. *Let $\langle \alpha_u, \gamma_u \rangle$ be a Galois connections from $\mathcal{Q}$ to $\mathcal{Q}'$ and $\langle \gamma_l, \alpha_l \rangle$ a Galois connection from $\mathcal{Q}'$ to $\mathcal{Q}$. Let $\langle \gamma'_l, \alpha'_l \rangle$ be a Galois connection between $\mathcal{Q}'$ and $\mathcal{Q}$ such that for all agents $p' \in \mathcal{Q}'$,*

$$\gamma_u(p') \preceq \gamma_l(p') \preceq \gamma'_l(p').$$

Then $\Psi = (\alpha_l, \alpha_u)$ and $\Psi' = (\alpha'_l, \alpha_u)$ are conservative approximations such that for all agents $p \in \mathcal{Q}$,

$$\alpha'_l(p) \preceq \alpha_l(p).$$

Given a Galois connection $\langle \alpha_u, \gamma_u \rangle$ between $\mathcal{Q}$ and $\mathcal{Q}'$, the tightest approximation is obtained using a Galois connection $\langle \gamma_l, \alpha_l \rangle$ from $\mathcal{Q}'$ to $\mathcal{Q}$ such that $\gamma_u = \gamma_l$. If no such connection can be found, then several "maximal" approximations may exist, but no tightest approximation. This is possible because γ_u is a concretization function and γ_l is an abstraction function of their respective Galois connections. These functions are not arbitrary, and must satisfy certain conditions that can be derived from the definition of Galois connection. While it is possible that some function γ_u satisfy both sets of conditions (which is necessary in order for γ_u to also be an abstraction map), this is not required.

A related result characterizes the inverse of a conservative approximation $\Psi = (\alpha_l, \alpha_u)$ induced by a pair of Galois connections. It shows that the inverse is defined at an agent $p' \in \mathcal{Q}'$ if and only if $\gamma_u(p')$ and $\gamma_l(p')$ are equal, and are "mutually" injective.

Theorem 6.3. *Let $\Psi = (\alpha_l, \alpha_u)$ be the conservative approximation from $\mathcal{Q}$ to $\mathcal{Q}'$ induced by the Galois connections $\langle \alpha_u, \gamma_u \rangle$ from $\mathcal{Q}$ to $\mathcal{Q}'$ and $\langle \gamma_l, \alpha_l \rangle$ from $\mathcal{Q}'$ to $\mathcal{Q}$. Then, for all agents $p' \in \mathcal{Q}'$, $\Psi_{inv}(p')$ is defined and is equal to p if and only if*

- $\gamma_u(p') = \gamma_l(p') = p$, *and*
- *if $p'_1 \in \mathcal{Q}'$ is an agent such that $\gamma_u(p'_1) = \gamma_l(p'_1) = p$, then $p'_1 = p'$.*

The conditions on γ_u and γ_l expressed in Theorem 6.3 are weaker than having γ_u and γ_l be injective. This is because the condition of injectivity (the second bullet above) must be satisfied only on those elements p'_1 for which *both* γ_u and γ_l evaluate to p. Because these results can be inverted to apply in the reverse direction (see Section 7 below), the weaker condition gives us a higher degree of freedom to construct abstractions.

Conversely, we can provide sufficient conditions for a conservative approximation to form a pair of Galois connections. It is sufficient that the upper and lower bound be monotonic (which is a necessary condition for Galois connections), and that the inverse of the conservative approximation be defined everywhere.

Theorem 6.4. *Let $\mathcal{Q}$ and $\mathcal{Q}'$ be models of computation and let $\Psi = (\Psi_l, \Psi_u)$ be a conservative approximation from $\mathcal{Q}$ to $\mathcal{Q}'$ such that*

1. Ψ_u and Ψ_l are monotonic, and
2. $\Psi_{inv}(p')$ is defined for all $p' \in \mathcal{Q}'.D$.

Then

- *$\langle \Psi_u, \Psi_{inv} \rangle$ is a Galois connection from $\mathcal{Q}.D$ to $\mathcal{Q}'.D$, and*
- *$\langle \Psi_{inv}, \Psi_l \rangle$ is a Galois connection from $\mathcal{Q}'.D$ to $\mathcal{Q}.D$.*

The condition that Ψ_{inv} be defined everywhere is crucial. In fact, there are monotonic conservative approximations such that the abstraction functions are not abstraction maps of any Galois connection. This occurs when the equivalence classes induced by Ψ_u and Ψ_l do not have the necessary greatest and lowest element, as described by Theorem 5.3.

The same relationship that holds between Galois connections and conservative approximations can be established between abstract interpretations and compositional conservative approximations. This correspondence follows from the fact that, despite being defined in different terms, the three conditions of Definition 4.1 are equivalent to the conditions S1 through S3 of Theorem 4.3. Therefore, abstract interpretations, when used in conjunction with a second Galois connection from $\mathcal{Q}'$ to $\mathcal{Q}$, induce *compositional* conservative approximations.

Corollary 6.5. *Let $\mathcal{Q}'$ be an abstract interpretation of $\mathcal{Q}$ by a Galois connection $\langle \alpha_u, \gamma_u \rangle$. Let $\langle \gamma_l, \alpha_l \rangle$ be Galois connection between $\mathcal{Q}'$ and $\mathcal{Q}$. Then the following two statements are equivalent:*

- *For all $p' \in \mathcal{Q}'.D$, $\gamma_u(p') \preceq \gamma_l(p')$.*
- *(α_l, α_u) is a compositional conservative approximation from $\mathcal{Q}$ to $\mathcal{Q}'$.*

The inverse of the conservative approximation is again characterized as already discussed in Corollary 6.2 and Theorem 6.3.

The relationship between the Galois connection and the conservative approximation from the continuous time model $\mathcal{Q}^C$ to the discrete time model $\mathcal{Q}^D$ derived in Section 3 can now be made precise. In particular, because the inverse of the conservative approximation is defined everywhere (Section 5), and because Ψ_l^C and Ψ_u^C are both monotonic, the pairs $\langle \Psi_u^C, \Psi_{inv}^C \rangle$ and $\langle \Psi_{inv}^C, \Psi_l^C \rangle$ are Galois connections. In addition, $\Psi_u^C = \alpha$ and $\Psi_{inv}^C = \gamma$. Therefore, $\langle \Psi_{inv}^C, \Psi_l^C \rangle$ is the additional Galois connection to be used in conjunction with $\langle \alpha, \gamma \rangle$ to obtain a conservative approximation. Our results also show that this approximation is the tightest that can be induced by $\langle \alpha, \gamma \rangle$.

7 Refinement and polymorphism

In Section 2.5 we have characterized a conservative abstraction as a pair of functions that form a conservative approximation. Similarly, a refinement can be established in the form of a conservative approximation that goes in the opposite direction. Thus, our notion of refinement does not correspond exactly to the inverse of the abstraction, since, as we have noted, the inverse may not be defined for all agents. Nonetheless, our results show that if the inverse *is* defined for some agent, then the upper and the lower bounds of the refinement are the same and are equal to the inverse.

In the rest of the paper, we will restrict our attention to conservative approximations induced by a pair of Galois connections. In fact, because abstraction and refinement are

symmetric, Galois connections are particularly well behaved and make it easy to derive the tight relationship that exists between the abstraction and the refinement functions. Observe, in fact, that in our previous results about Galois connections, the hypothesis were symmetric relative to our domains of agents: a Galois connection exists from $\mathcal{Q}$ to $\mathcal{Q}'$, and a second Galois connection exists from $\mathcal{Q}'$ to $\mathcal{Q}$. Thus, those results are dually valid by simply replacing all occurrences of α_u by γ_l, and all occurrences of γ_u by α_l, and by exchanging the domains of agents. Therefore, the same pair of Galois connections may induce two conservative approximations, one from $\mathcal{Q}$ to $\mathcal{Q}'$ (the abstraction), and a second from $\mathcal{Q}'$ to $\mathcal{Q}$ (the refinement). As a special case, if $\mathcal{Q}$ is a refinement of $\mathcal{Q}'$, in the sense that it implements all of the agents in $\mathcal{Q}'$, then the conservative approximation from $\mathcal{Q}'$ to $\mathcal{Q}$ is *not* a proper abstraction.

Suppose now that $\mathcal{Q}$ and $\mathcal{Q}'$ are models, and that $\Psi = (\alpha_l, \alpha_u)$ is a conservative approximation from $\mathcal{Q}$ to $\mathcal{Q}'$ induced by a pair of Galois connections $\langle \alpha_u, \gamma_u \rangle$ and $\langle \gamma_l, \alpha_l \rangle$. Theorem 6.1 shows that in order for $\Psi' = (\gamma_u, \gamma_l)$ to be a conservative approximation from $\mathcal{Q}'$ to $\mathcal{Q}$ we need that $\alpha_l(p) \preceq \alpha_u(p)$ for all agents $p \in \mathcal{Q}.D$. This condition is commonly satisfied by a conservative approximation Ψ, and simply formalizes the intuition that the lower bound of an agent must be less than or equal to its upper bound (although, as noted earlier, this is not a necessary condition for a conservative approximation). In addition, the inverses Ψ_{inv} and Ψ'_{inv} of the conservative approximations are inverse of each other, i.e., for all $p \in \mathcal{Q}.D$ and $p' \in \mathcal{Q}'.D$, $\Psi_{inv}(p') = p$ if and only if $\Psi'_{inv}(p) = p'$. The situation is therefore the one depicted in Fig. 1, where Galois connections are denoted by pairs of dotted arcs and by a straight arrow that indicates the direction of the connection. The shaded region in $\mathcal{Q}$ corresponds to the set of agents that can be represented exactly in $\mathcal{Q}'$. This region is isomorphic to the corresponding shaded region in $\mathcal{Q}'$ which consists of the agents in $\mathcal{Q}'$ that can be represented exactly in $\mathcal{Q}$. In other words, a subset of the agents of the two semantic domains can be represented indifferently in either domain, while the remaining agents can only be approximated by the other domain (i.e., their upper and lower bounds do not coincide; see Section 5, Definition 5.2). We say that an agent p that can be represented exactly in two

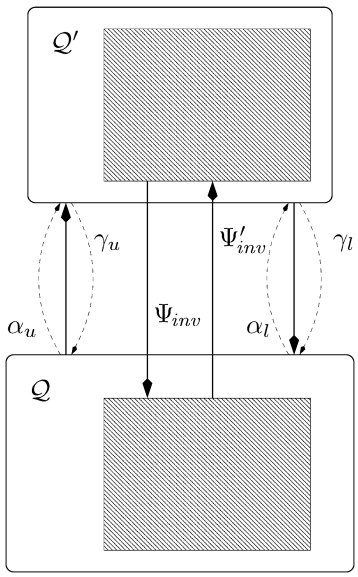

Fig. 1 Abstraction, Refinement and their Inverses

different models of computation $\mathcal{Q}$ and $\mathcal{Q}'$ is *polymorphic relative to* $\mathcal{Q}$ *and* $\mathcal{Q}'$. In this case, while the form in which the agent is represented in the two models may be different, its behavior is such that it does not rely on any assumption that can be expressed in only one of the models. This notion of polymorphism depends upon the particular choice of conservative approximation between the agent models. This is because *the approximation* determines how each model interprets the agents that belong to the other model. In particular, the approximation determines what information is and is not expressible in one model, relative to the other. Different approximations may thus lead to different forms of interpretations. This is unlike other notions of polymorphism that rely upon a common underlying semantics, such as the automata model in Ptolemy II [19].

If $\mathcal{Q}'$ is strictly more abstract than $\mathcal{Q}$, in the sense that the agents in $\mathcal{Q}'$ contain strictly less information than those in $\mathcal{Q}$, then Ψ_{inv} is total (assuming Ψ is the tightest conservative approximation). In that case, the conservative approximation from $\mathcal{Q}'$ to $\mathcal{Q}$ is essentially an embedding of $\mathcal{Q}'.D$ into $\mathcal{Q}.D$, equipped with the respective orders, and the shaded region in $\mathcal{Q}'$ extends to the entire domain of $\mathcal{Q}'$. Hence, all agents of $\mathcal{Q}'$ can be represented exactly in $\mathcal{Q}$, and it is therefore straightforward to consider the heterogeneous composition of agents in the context of $\mathcal{Q}$. This is the case, for example, of our continuous and discrete time models $\mathcal{Q}^C$ and $\mathcal{Q}^D$. There, as already discussed, a continuous time agent is polymorphic whenever the actions that it executes occur non-deterministically in an integer time interval.

The composition of non-polymorphic agents across model boundaries is more problematic. Assume in particular that $p \in \mathcal{Q}.D$ is an agent such that $\Psi_u(p) \neq \Psi_l(p)$. In that case, p is not represented exactly in $\mathcal{Q}'$, or, to put it another way, p is not polymorphic relative to the chosen domains. There are different ways to get around this problem, and they mainly consist of encapsulating p using a translator that *does* make the combination polymorphic. This is, for example, the technique used in the Ptolemy II framework, where an intermediate director compatible with the agent is used to mediate the communication between the agent that is not polymorphic, and the domain in which the designer wishes to use it.

Translations in our framework take the form of the topological notions of *closure* and *interior* systems [26]. We have already seen that a conservative approximation $\Psi = (\Psi_l, \Psi_u)$ determines for each agent the equivalence classes of the agents that have the same upper bound and the same lower bound, respectively. Theorem 5.3 shows that if the inverse of a conservative approximation is defined for an agent p', then $\Psi_{inv}(p')$ is the greatest and least element, respectively, of these equivalence classes. When the upper and lower bound are monotonic functions, these elements constitute a closure and an interior for the elements of their respective equivalence classes.

Theorem 7.1. *Let* $\Psi = (\Psi_l, \Psi_u)$ *be a conservative approximation from* $\mathcal{Q}$ *to* $\mathcal{Q}'$ *such that* Ψ_l *and* Ψ_u *are monotonic and* Ψ_{inv} *is defined for all agents* p' *in* $\mathcal{Q}'$. *Let* $C, I : \mathcal{Q}.D \mapsto \mathcal{Q}.D$ *be operators of* $\mathcal{Q}$ *defined as*

$$C(p) = \Psi_{inv}(\Psi_u(p)),$$
$$I(p) = \Psi_{inv}(\Psi_l(p)).$$

Then C *is a closure operator, and* I *is an interior operator.*

The closure and the interior operator essentially "complete" an agent in order to make it compatible with the requirements of the abstract domain. The closure produces an abstraction within $\mathcal{Q}$ by choosing the greatest element of the equivalence class induced by Ψ_u, thus

potentially "adding" behaviors that are required by the abstract domain. The interior, on the other hand, computes a refinement in $\mathcal{Q}$, by choosing the least element of the equivalence class induced by Ψ_I, and thus "removing" behaviors that are incompatible with the abstract domain. Other forms of completion that, for example, handle the case in which Ψ_{inv} is not total are also possible. We do not however explore them further here, and reserve them for our future work. Our current interest is also directed towards understanding the relationships between the operational semantics of simulators for different models of computation, and to derive synchronization constraints that make the simulation consistent with the abstraction. Our approach consists of augmenting the models with partial behaviors that denotationally represent the progress in terms of simulation steps.

8 Related work

In addition to abstract interpretations and conservative approximations, other forms of abstraction and methods for heterogeneous modeling and for verification have been proposed in the literature.

One example is the *homomorphic reduction* proposed by Kurshan [15, 16]. This technique can be applied to models of behavior that consist of languages (sets of sequences) that are recognized by a class of automata. Here, each automaton P constructed over a set of symbols L (an L-automaton) accepts a language $\mathcal{L}(P) \subseteq L^\omega$, where L^ω denotes the set of all infinite sequences of symbols from L. Verification in this context is the process of determining whether the language $\mathcal{L}(P)$ recognized by an implementation automaton P is contained in the language $\mathcal{L}(T)$ accepted by the specification automaton T, i.e., that $\mathcal{L}(P) \subseteq \mathcal{L}(T)$. This problem can be reduced to a more abstract language L' by verifying that $\mathcal{L}(P') \subseteq \mathcal{L}(T')$, for appropriate abstract L'-automata P' and T'. The main result[1] states that $\mathcal{L}(P') \subseteq \mathcal{L}(T')$ implies $\mathcal{L}(P) \subseteq \mathcal{L}(T)$ provided there exists a *language homomorphism* $\Phi : L^\omega \mapsto L'^\omega$ such that $\Phi(\mathcal{L}(P)) \subseteq \mathcal{L}(P')$ and $\Phi(\mathcal{L}(T^\#)) \subseteq \mathcal{L}(T'^\#)$. In this case, Φ is said to be *co-linear*[2] for $(P, T; P', T')$. In the co-linearity condition above, the notation $T^\#$ denotes the *dual* automaton[3] of T, which is closely related to language complementation.

In Lemma 2.2 we have argued that one function on languages is not sufficient to guarantee the preservation of such verification result. The apparent contradiction with the use of just one language homomorphism Φ can be reconciled by accounting for the use of the dual automaton in the co-linearity condition. Effectively, if Φ is co-linear for $(P, T; P', T')$, then it can be shown that not only is $\Phi(\mathcal{L}(P)) \subseteq \mathcal{L}(P')$, but also that $\mathcal{L}(T') \subseteq \Phi(\overline{\mathcal{L}(T)})$, where the overline bar denotes language complementation. Hence, the language of the specification T is transformed according to a different abstraction functions, namely $\Theta(\mathcal{L}(T)) = \overline{\Phi(\overline{\mathcal{L}(T)})}$. Interestingly, Φ and Θ form the upper and lower bound of a conservative approximation that is closely related (and under certain conditions equal) to the conservative approximation induced by a homomorphism [3, 4, 24]. Co-linearity of Φ thus simply ensures that $\mathcal{L}(P')$ and $\mathcal{L}(T')$ provide looser bounds, a condition that still guarantees soundness in the verification. Conservative approximations generalize the technique of homomorphic reduction to arbitrary agent models, and can therefore be applied to models that are not described by automata.

[1] Theorem 8.5.2 in [16].

[2] Definition 8.5.1 in [16].

[3] Definitions 6.2.19 and 6.2.26 in [16].

Model checking techniques based on abstraction/refinement is also a well studied related field of application for abstraction mappings [1, 7, 10], and is a typical application of the framework of abstract interpretations. The technique consists of first deriving an over-approximation of a state-based model using, for instance, predicate abstraction [13], in which each state is characterized by the predicates that it satisfies. A transitions exists between two abstract states whenever a transition exists between any of the possible corresponding concrete states. The abstract model is constructed in a way that ensures that the property to be verified can be represented exactly (by, for example, an appropriate choice of the predicates). Therefore, if the property is verified in the abstract domain, it is also verified in the concrete domain (see Theorem 2.4). If not, an abstract counter-example is generated and compared to the concrete model. If a corresponding concrete counter-example exists, then the property is violated. Otherwise, the abstract domain is refined by introducing additional predicates in a way that rules out the abstract counter-example, and the procedure is repeated until the satisfaction of the property is determined. The approach based on conservative approximations differs because, as explained, it allows non-trivial abstraction of the specification, as well as of the implementation. Exploring the ramifications of our technique in the area of model checking is, however, part of our future work. Model checking techniques also exist that use under-approximations, rather than over-approximations, to derive an abstracted model [23]. This is similar to our use of the lower bound function. However, unlike our use of the lower bound, the under-approximation is applied to the implementation, rather than to the specification. This corresponds again to using a Galois connection, one that goes in the reverse direction. By doing this, if the abstract model violates the property under verification, then it can be concluded that also the concrete model violates the property. Instead, if the abstract model satisfies the property, the verification is inconclusive and the abstract model must be refined until the property is proved incorrect, or the abstraction becomes exact. This approach may be useful when the interest lies in finding true counter-examples and bug traces. One could apply a similar technique using conservative approximations, where the lower bound is applied to the implementation and the upper bound to the specification. In this case, the negative refinement result would be preserved from the abstract to the concrete model. This is also part of our future work.

Another formalization of abstraction is based on *theory interpretations* [21]. Here, an abstract architecture description and a concrete architecture description are both translated to theories in a logical language (typically first-order logic). The concrete architecture is correct relative to the abstract architecture if there is a theory interpretation I from the abstract theory Θ to the concrete theory Θ'; that is, for every formula F,

$$F \in \Theta \Rightarrow I(F) \in \Theta'.$$

In addition, it may be required that I be faithful:

$$F \notin \Theta \Rightarrow I(F) \notin \Theta'.$$

Our approach does not interpret architectures, or other agents, as logical theories. Instead, they are directly modeled as mathematical objects. This can be thought of as a model based approach, as opposed to a theory based approach. In a model based approach, within a given model of computation, the refinement relation is just a binary relation on objects in the model. This notion of refinement is easier to reason about than theory interpretations, but it is less flexible for comparing agents in different models of computation. This can be addressed by

introducing abstract interpretations or conservative approximations. A major contribution of this paper is new insights into the relationship between these model based approaches.

Process Spaces [22] is a very general class of concurrency models, and it compares quite closely to trace-based agent models [24]. Given a set of executions $\mathcal{E}$, a Process Space $\mathcal{S}_\mathcal{E}$ consists of the set of all the processes (X, Y), where X and Y are subsets of $\mathcal{E}$ such that $X \cup Y = \mathcal{E}$. The sets of executions X and Y of a process are not necessarily disjoint, and they represent the assumptions (Y) and the guarantees (X) of the process with respect to its environment. As in trace-based agent models, executions are abstract objects.

Different sets of abstract executions $\mathcal{E}_1$ and $\mathcal{E}_2$ induce different Process Spaces $\mathcal{S}_{\mathcal{E}_1}$ and $\mathcal{S}_{\mathcal{E}_2}$. The notion of process abstraction from $\mathcal{S}_{\mathcal{E}_1}$ to $\mathcal{S}_{\mathcal{E}_2}$ in Process Spaces is related to the notion of conservative approximation. In particular, process abstractions are defined as the Galois connections between process spaces that are derived from a relation on the set of abstract executions. The connections are obtained as axialities [12]. A process abstraction is classified as optimistic or pessimistic according to whether it preserves certain verification results from the concrete to the abstract or from the abstract to the concrete model. These two kinds of abstraction can be used in combination to preserve verification results both ways. However, in that case, the two models are isomorphic since there is effectively no loss information. Optimistic and pessimistic process abstractions roughly correspond to the two abstraction functions of conservative approximations. However, our use of these functions is significantly different, since we apply them in combination (one for the specification, the other for the implementation). Consequently, even when used in combination, our models need not be isomorphic, so that we obtain stronger preservation results without sacrificing the abstraction.

Winskel et al. [25] propose a framework based on category theory that is related to ours. In their formalism, each model of computation is turned into a category where the objects are the agents, and the morphisms represent a refinement relationship based on *simulations* between the agents. The authors study a variety of different models that are obtained by selecting arbitrary combinations of three parameters: behavior vs. system (e.g., traces vs. state machines), interleaving vs. non-interleaving (e.g., state machines vs. event structures) and linear vs. branching time. The common operations in a model are derived as universal constructions in the category. Relationships can be constructed by relating the categories corresponding to different models by means of functors, which are homomorphisms of categories that preserve morphisms and their compositions. When categories represent models of computation, functors establish connections between the models in a way similar to abstraction maps and semantic functions. In particular, when the morphisms in the category are interpreted as refinement, functors become essentially monotonic functions between the models, since preserving morphisms is equivalent to preserving the refinement relationship.

In [25], the authors thoroughly study the relationships between the eight different models of concurrency above by relating the corresponding categories through functors. In addition, these functors are shown to be components of *reflections* or *co-reflections*. These are particular kinds of adjoints, which are pairs of functors that go in opposite directions and enjoy properties that are similar to the order preservation of the abstraction and concretization maps of a Galois connection. When the morphisms are interpreted as refinement, reflections and co-reflections generalize the concept of Galois connection to preorders. In fact, the relationships between categories based on adjoints are similar in nature to the abstractions and refinements obtained by abstract interpretations and conservative approximations. However, as described above for abstract interpretations, conservative approximations use independent abstractions for the implementation and the specification in order to derive a stronger result in terms of

preservation of the refinement relation, and avoidance of false positive verification results. Indeed, we require two Galois connections, instead of one, to determine a single conservative approximation (see Section 6). In the work presented in [25], this translates in two adjoints per pair of categories.

9 Conclusions

We presented the use of abstraction and refinement functions between models of computation for the verification and design of heterogeneous systems. We compared conservative approximations to abstract interpretations and we showed that, unlike abstract interpretations, conservative approximations always preserve refinement verification results from an abstract to a concrete model, while avoiding false positives. Therefore, conservative approximations are better suited for design methodologies that use several models of computation. In particular, because they always guarantee correctness, conservative approximations provide more flexibility in choosing the verification strategy and the hierarchy of models used in the design flow. We then completed our comparison by showing that a class of conservative approximations can be derived by using *pairs* of abstract interpretations (or Galois connections), going in opposite directions. A set of necessary and sufficient conditions characterizes the abstraction and concretization functions that can be used for this purpose. In addition, we introduced the inverse of a conservative approximation to identify components that can be used indifferently in several models, thus enabling reuse across domains of computation. By providing embeddings between agent models, inverses induce topological closure and interior operators that identify ways of adapting agent behaviors when interfacing between different models. The resulting theory can be used as the basis of frameworks that support heterogeneous modeling.

Our current work focuses on extending techniques that make it easier to construct conservative approximations between agent models. The axialities of homomorphisms on behaviors described in this paper is one such example. However, homomorphisms are usually defined to preserve the alphabet of behaviors, so that the induced conservative approximations, too, must preserve the alphabet of agents. More interesting conservative approximations can be constructed by letting the homomorphism change the alphabet of a behavior, for example by hiding certain signals, like clocks and activation signals, that have no meaning in a more abstract model. This is also appropriate for converting a detailed protocol specification into a more abstract, transaction-based, specification. Arbitrary changes of the alphabet are also possible. In this case, however, the homomorphism must not only be applied to the behaviors, but also to the operators, in order to correctly translate expressions. In this case the homomorphism becomes similar to a functor between categories, where a category has behaviors as objects and the operators as morphisms.

A model that uses behaviors as its underlying structure may impose restrictions on the kind of agents that can be constructed. For example, only receptive (or progressive, or input enabled) agents might be allowed. The axialities of a homomorphism, however, may not necessarily yield agents that satisfy such conditions. A promising avenue of future research consists therefore in identifying the agent that most faithfully approximates the missing abstraction, while satisfying the constraints imposed by the model, and while still functioning as the bound of a conservative approximation. This would constitute a generalization of the technique proposed by Loiseaux et al. [20] on property-preserving abstractions in the context of transition systems.

A Appendix: Proofs of main results

A.1 Proof of Theorem 6.1

The proof of Theorem 6.1 relies on the following lemma.

Lemma A.1. *Let $\mathcal{Q}$ and $\mathcal{Q}'$ be models of computation and let $\langle \alpha_u, \gamma_u \rangle$ be a Galois connections from $\mathcal{Q}$ to $\mathcal{Q}'$ and $\langle \gamma_l, \alpha_l \rangle$ a Galois connection from $\mathcal{Q}'$ to $\mathcal{Q}$. Then the following two statements are equivalent:*

1. For all agents $p' \in \mathcal{Q}'$, $\gamma_u(p') \preceq \gamma_l(p')$.
2. For all agents p_1 and p_2 in $\mathcal{Q}$, $\alpha_u(p_1) \preceq \alpha_l(p_2) \Rightarrow p_1 \preceq p_2$.

Proof: For the forward direction ($1 \Rightarrow 2$), let p_1 and p_2 be agents from $\mathcal{Q}$, and assume $\alpha_u(p_1) \preceq \alpha_l(p_2)$. Since $\langle \alpha_u, \gamma_u \rangle$ is a Galois connection, by Theorem 2.3, $p_1 \preceq \gamma_u(\alpha_u(p_1))$. The proof is then completed by the following series of implications.

$$
\begin{aligned}
& p_1 \preceq \gamma_u(\alpha_u(p_1)) \\
& \quad \text{since by hypothesis } \alpha_u(p_1) \preceq \alpha_l(p_2), \\
& \quad \text{and since, by Theorem 2.3, } \gamma_u \text{ is monotonic} \\
& \Rightarrow p_1 \preceq \gamma_u(\alpha_l(p_2)) \\
& \quad \text{since by hypothesis, } \gamma_u \preceq \gamma_l \\
& \Rightarrow p_1 \preceq \gamma_l(\alpha_l(p_2)) \\
& \quad \text{since, by Theorem 2.3, } \gamma_l(\alpha_l(p_2)) \preceq p_2, \text{ and by transitivity} \\
& \Rightarrow p_1 \preceq p_2.
\end{aligned}
$$

For the reverse direction ($2 \Rightarrow 1$), let $p' \in \mathcal{Q}'$ be an agent. By Theorem 2.3, $\alpha_u(\gamma_u(p')) \preceq p'$ and $p' \preceq \alpha_l(\gamma_l(p'))$. Therefore, by transitivity, $\alpha_u(\gamma_u(p')) \preceq \alpha_l(\gamma_l(p'))$ and consequently, by hypothesis, $\gamma_u(p') \preceq \gamma_l(p')$. □

Theorem 6.1 follows directly from the definition of conservative approximation and from Lemma A.1.

A.2 Proof of Theorem 6.3

The proof of Theorem 6.3 relies on the following lemma.

Lemma A.2. *Let $\mathcal{Q}$ and $\mathcal{Q}'$ be models of computation and let $\langle \alpha_u, \gamma_u \rangle$ be a Galois connections from $\mathcal{Q}$ to $\mathcal{Q}'$ and $\langle \gamma_l, \alpha_l \rangle$ a Galois connection from $\mathcal{Q}'$ to $\mathcal{Q}$ such that for all agents $p' \in \mathcal{Q}'$, $\gamma_u(p') \preceq \gamma_l(p')$. Then for all agents $p \in \mathcal{Q}$ and $p' \in \mathcal{Q}'$ the following two statements are equivalent:*

1. $\alpha_u(p) = \alpha_l(p) = p'$
2. • *$\gamma_u(p') = \gamma_l(p') = p$, and*
  • *if $p'_1 \in \mathcal{Q}'$ is an agent such that $\gamma_u(p'_1) = \gamma_l(p'_1) = p$, then $p'_1 = p'$.*

Proof: For the forward direction (1 $\Rightarrow$ 2), let p be an agent from $\mathcal{Q}$, and assume $\alpha_u(p) = \alpha_l(p) = p'$. Since $\langle \alpha_u, \gamma_u \rangle$ is a Galois connection, by Theorem 2.3, $p \preceq \gamma_u(\alpha_u(p))$. Consider the following series of implications.

$$
\begin{aligned}
& p \preceq \gamma_u(\alpha_u(p)) \\
& \quad \text{since by hypothesis } \alpha_u(p) = \alpha_l(p) \\
& \Rightarrow p \preceq \gamma_u(\alpha_u(p)) = \gamma_u(\alpha_l(p)) \\
& \quad \text{since by hypothesis, } \gamma_u \preceq \gamma_l \\
& \Rightarrow p \preceq \gamma_u(\alpha_u(p)) = \gamma_u(\alpha_l(p)) \preceq \gamma_l(\alpha_l(p)) \\
& \quad \text{by Theorem 2.3} \\
& \Rightarrow p \preceq \gamma_u(\alpha_u(p)) \preceq \gamma_l(\alpha_l(p)) \preceq p \\
& \quad \text{since by hypothesis } \alpha_u(p) = \alpha_l(p) = p' \\
& \Rightarrow p \preceq \gamma_u(p') \preceq \gamma_l(p') \preceq p.
\end{aligned}
$$

Therefore $\gamma_u(p') = \gamma_l(p') = p$. Let now $p'_1 \in \mathcal{Q}'$ be such that $\gamma_u(p'_1) = \gamma_l(p'_1) = p$. Then, since $\langle \alpha_u, \gamma_u \rangle$ and $\langle \alpha_l, \gamma_l \rangle$ are Galois connections, and since by hypothesis $p \preceq \gamma_u(p'_1)$ and $\gamma_l(p'_1) \preceq p$,

$$\alpha_u(p) \preceq p'_1 \preceq \alpha_l(p).$$

Since by hypothesis $\alpha_u(p) = \alpha_l(p) = p'$, $p' \preceq p'_1 \preceq p'$. Therefore, $p'_1 = p'$.

For the reverse direction (2 $\Rightarrow$ 1), let $p' \in \mathcal{Q}'$ be an agent such that $\gamma_u(p') = \gamma_l(p') = p$, and assume that if $p'_1 \in \mathcal{Q}'$ is such that $\gamma_u(p'_1) = \gamma_l(p_1) = p$ then $p'_1 = p'$. Note that because $p = \gamma_u(p')$, it is also $p \preceq \gamma_u(p')$, and therefore, since $\langle \alpha_u, \gamma_u \rangle$ is a Galois connection, $\alpha_u(p) \preceq p'$. Then, consider the following series of implications that start from the result of Theorem 2.3:

$$
\begin{aligned}
& p \preceq \gamma_u(\alpha_u(p)) \\
& \quad \text{since by hypothesis, } \gamma_u \preceq \gamma_l \\
& \Rightarrow p \preceq \gamma_u(\alpha_u(p)) \preceq \gamma_l(\alpha_u(p)) \\
& \quad \text{since by the argument above } \alpha_u(p) \preceq p' \\
& \quad \text{and since } \gamma_l \text{ is monotonic (by Theorem 2.3)} \\
& \Rightarrow p \preceq \gamma_u(\alpha_u(p)) \preceq \gamma_l(\alpha_u(p)) \preceq \gamma_l(p') \\
& \quad \text{since by hypothesis } \gamma_l(p') = p \\
& \Rightarrow p \preceq \gamma_u(\alpha_u(p)) \preceq \gamma_l(\alpha_u(p)) \preceq \gamma_l(p') = p.
\end{aligned}
$$

Therefore, $\gamma_u(\alpha_u(p)) = \gamma_l(\alpha_u(p)) = p$. Hence, by hypothesis, $\alpha_u(p) = p'$. The proof that $\alpha_l(p) = p'$ is similar. □

Theorem 6.3 follows directly from the definition of inverse of a conservative approximation and from Lemma A.2.

A.3 Proof of Theorem 6.4

Theorem 6.4. *Let $\mathcal{Q}$ and $\mathcal{Q}'$ be models of computation and let $\Psi = (\Psi_l, \Psi_u)$ be a conservative approximation from $\mathcal{Q}$ to $\mathcal{Q}'$ such that*

1. *Ψ_u and Ψ_l are monotonic, and*
2. *$\Psi_{inv}(p')$ is defined for all $p' \in \mathcal{Q}'$.*

Then

- *$\langle \Psi_u, \Psi_{inv} \rangle$ is a Galois connection from $\mathcal{Q}$ to $\mathcal{Q}'$, and*
- *$\langle \Psi_{inv}, \Psi_l \rangle$ is a Galois connection from $\mathcal{Q}'$ to $\mathcal{Q}$.*

Proof: We show that $\langle \Psi_u, \Psi_{inv} \rangle$ is a Galois connection by proving that for all agents $p \in \mathcal{Q}$ and $p' \in \mathcal{Q}'$,

$$\Psi_u(p) \preceq p' \Leftrightarrow p \preceq \Psi_{inv}(p').$$

We separately prove the forward and backward implications as follows.

$$\begin{aligned}
&\Psi_u(p) \preceq p' \\
&\qquad \text{by definition of inverse of a conservative approximation} \\
&\Leftrightarrow \Psi_u(p) \preceq \Psi_l(\Psi_{inv}(p')) \\
&\qquad \text{and since } (\Psi_l, \Psi_u) \text{ is a conservative approximation} \\
&\Rightarrow p \preceq \Psi_{inv}(p').
\end{aligned}$$

Similarly,

$$\begin{aligned}
&p \preceq \Psi_{inv}(p') \\
&\qquad \text{since, by hypothesis, } \Psi_u \text{ is monotonic} \\
&\Rightarrow \Psi_u(p) \preceq \Psi_u(\Psi_{inv}(p')) \\
&\qquad \text{by definition of inverse of a conservative approximation} \\
&\Leftrightarrow \Psi_u(p) \preceq p'.
\end{aligned}$$

The proof that $\langle \Psi_{inv}, \Psi_l \rangle$ is a Galois connection is similar. □

References

1. Alur R, Itai A, Kurshan R, Yannakakis M (1995) Timing verification by successive approximation. Inf Comput 118(1):142–157
2. Balarin F, Lavagno L, Passerone C, Sangiovanni-Vincentelli A, Watanabe Y, Yang G (2002) Concurrent execution semantics and sequential simulation algorithms for the metropolis meta-model. In: Proceedings of the tenth international symposium on hardware/software codesign. Estes Park, CO, May 2002
3. Burch JR (1992) Trace algebra for automatic verification of real-time concurrent systems. PhD thesis, School of Computer Science, Carnegie Mellon University

4. Burch JR, Passerone R, Sangiovanni-Vincentelli AL (2001) Overcoming heterophobia: modeling concurrency in heterogeneous systems. In: Koutny M, Yakovlev A (eds) Application of concurrency to system design
5. Burch JR, Passerone R, Sangiovanni-Vincentelli AL (2001) Using multiple levels of abstractions in embedded software design. In: Henzinger and Kirsch [14], pp 324–343
6. Burch JR, Passerone R, Sangiovanni-Vincentelli AL (2002) Modeling techniques in design-by-refinement methodologies. In: Proceedings of the sixth biennial world conference on integrated design and process technology
7. Clarke EM, Grumberg O, Peled D (1999) Model checking, 2nd edn. The MIT Press, Cambridge, MA
8. Cousot P, Cousot R (1977) Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: Conference record of the fourth annual ACM SIGPLAN-SIGACT symposium on principles of programming languages. Los Angeles, California. ACM Press, New York, NY, pp 238–252
9. Cousot P, Cousot R (1992) Comparing the Galois connection and widening/narrowing approaches to abstract interpretation, invited paper. In: Bruynooghe M, Wirsing M (eds) Proceedings of the international workshop programming language implementation and logic programming, PLILP'92, Leuven, Belgium, Lecture notes in computer science, volume 631. Springer-Verlag, Berlin, Germany, pp 269–295
10. Das S, Dill DL (2001) Successive approximation of abstract transition relations. In: Proceedings of the sixteenth annual IEEE symposium on logic in computer science. Boston, MA
11. Dill DL (1989) Trace theory for automatic hierarchical verification of speed-independent circuits. ACM Distinguished Dissertations. MIT Press
12. Erné M, Koslowski J, Melton A, Strecker GE (1993) A primer on galois connections. In: Papers on general topology and applications, volume 704 of Ann. New Yosk Acad. Sci. Madison, WI, pp 103–125
13. Graf S, Saidi H (1997) Construction of abstract state graphs with PVS. In: Computer-aided verification, proceedings of the 1997 workshop, volume 1254 of Lectures notes in computer science
14. Henzinger TA, Kirsch CM (eds) (2001) Embedded software, volume 2211 of Lecture notes in computer science. Springer-Verlag
15. Kurshan RP, McMillan KL (1991) Analysis of digital circuits through symbolic reduction. IEEE Trans Comput-Aided Design Integr Circuits 10(11):1356–1371
16. Kurshan RP (1995) Computer-aided verification of coordinating processes: the automata-theoretic approach. Princeton University Press
17. Lee EA, Sangiovanni-Vincentelli AL (1998) A framework for comparing models of computation. IEEE Trans Comput-Aided Design Integr Circuits 17(12):1217–1229
18. Lee EA (2003) Overview of the Ptolemy project. Technical memorandum UCB/ERL M03/25. University of California, Berkeley
19. Lee EA, Xiong Y (2001) System-level types for component-based design. In: Henzinger and Kirsch [14]
20. Loiseaux C, Graf S, Sifakis J, Bouajjani A, Bensalem S (1995) Property preserving abstractions for the verification of concurrent systems. Formal Methods Syst Des 6:1–35
21. Moriconi M, Qian X, Riemenschneider RA (1995) Correct architecture refinement. IEEE Trans Softw Eng 21(4):356–372
22. Negulescu R (1998) Process spaces and the formal verification of asynchronous circuits. PhD thesis, University of Waterloo, Canada
23. Pasareanu C, Pelánek R, Visser W (2005) Concrete model checking with abstract matching and refinement. In: Proceedings of the 17th international conference on computer-aided verification, volume 3576 of Lecture notes in computer science. Springer-Verlag
24. Passerone R (2004) Semantic foundations for heterogeneous systems. PhD thesis, Department of EECS, University of California at Berkeley, 2004
25. Sassone V, Nielsen M, Winskel G (1996) Models for concurrency: towards a classification. Theor Comput Sci 170:297–348
26. Sutherland WA (1975) Introduction to metric and topological spaces. Oxford University Press, London, UK