Conservative Approximations for Heterogeneous Design

Roberto Passerone
Cadence Design Systems, Inc.
Berkeley CA 94704
robp@cadence.com

Jerry R. Burch
Synopsys, Inc.
Hillsboro OR 97124
jrb@synopsys.com

Alberto L. Sangiovanni-Vincentelli
Department of EECS
University of California
Berkeley CA 94720
alberto@eecs.berkeley.edu

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 system and their collective behavior as a *model of computation*. In this paper, we focus in particular on abstraction and refinement relationships in the form of *conservative approximations*. We do so by constructing a framework, called Agent Algebra, where the different models reside and share a common algebraic structure. We compare our techniques to the well established notion of abstract interpretation. We show that, unlike abstract interpretations, conservative approximations preserve refinement verification results from an abstract to a concrete model while avoiding false positives. 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 domains of computation.

Categories and Subject Descriptors

F.1.1 [**Computation by Abstract Devices**]: Models of Computation—*Relations between models*; F.3.2 [**Logics and Meanings of Programs**]: Semantics of Programming Languages—*Algebraic approaches to semantics*, *Denotational semantics*; I.6.4 [**Simulation and Modeling**]: Model Validation and Analysis

General Terms

Design, Theory, Verification

Keywords

Approximation, Abstraction, Refinement, Verification, Model of computation, Heterogeneous, Polymorphism

EMSOFT'04, September 27–29, 2004, Pisa, Italy.

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 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. 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.

Currently deployed design methodologies for heterogeneous embedded systems are often based on *ad hoc* techniques that lack formal foundations and hence are 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. We informally refer to the notation and the rules that are used to specify and verify the elements of a system and their collective behavior as a *model of computation*. The objective of this work is to provide a formal framework to uniformly present and reason about the characteristics and the properties of the different models of computation used in a design, and about their relationships. We accomplish this by defining an algebra that consists of the set of the denotations, called the *agents*, of the elements of a model and of the main operations that can be performed to compose agents and obtain a new agent. Different models of computation are still constructed as distinct algebras in our framework. However, we can take advantage of the common algebraic structure to derive results that apply to all models in the framework, and to relate different models using structure-preserving maps. In this paper, we focus in particular on abstraction and refinement relationships in the form of *conservative approximations*, and compare them to the well established notion of abstract interpretation [2]. We show that, unlike abstract interpretations, conservative approximations preserve refinement verification results from an abstract to a concrete model while avoiding false positives. 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 domains of computation.

We first present an account of the related work. Then, in section 2, we present the framework of Agent Algebra, followed in section 3 by the notion of conservative approximation. Section 4 compares conservative approximations and abstract interpretations. Finally, in section 5 we use conservative approximations to define a notion of polymorphism between agent models. An example of continuous and discrete time models is used throughout this work. The proofs for the results stated in this paper can be found in [12].

1.1 Related Work

The concept of a conservative approximation in our framework is derived from the one introduced by Burch [1]. In our work we generalize his approach and apply it to an algebra of arbitrary agents, rather than of arbitrary executions. Here we decompose the definition of conservative approximation to highlight and discuss its compositionality properties, and study its relationship with traditional notions of abstraction, such as Galois connections and abstract interpretations. In addition we further characterize the inverse of a conservative approximation, and use it to identify interactions between different models of computation.

Abstract interpretations are a widely used means of relating different domains of computation for the purpose of facilitating the analysis of a system [2, 3]. 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 presumably computationally more efficient.

Our notion of conservative approximation is closely related to that of an abstract interpretation, and a detailed account of the similarities and differences is presented in section 4. In particular, the upper bound of a conservative approximation and the inverse of the conservative approximation form, in some cases, a Galois connection and/or an abstract interpretation. However, the lower bound of a conservative approximation does not have an analogue in the theory of abstract interpretations. Nonetheless, in section 4 we show that the lower bound of a conservative approximation can be explained as the concretization map of another Galois connection, one that goes from the abstract to the concrete model. A conservative approximation is thus composed of *two pairs* of related functions, instead of just one, and are used in combination to derive stronger preservation results. In particular, by applying one pair to the implementation and the other to the specification, we are able to not only guarantee that certain properties are preserved from the abstract to the concrete domain, but also that a refinement verification result is preserved in the same direction. To our knowledge, for abstract interpretations a positive refinement verification result in the abstract domain implies a positive verification result in the concrete domain only if there is no loss of information when mapping the specification from the concrete domain to the abstract domain. Thus, conservative approximations allow non-trivial abstraction of both the implementation and the specification, while abstract interpretations only allow non-trivial abstraction of the implementation.

Process Spaces [11] is a very general class of concurrency models, and it compares quite closely to trace-based agent models [12]. 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 axialities of a relation and its inverse on the set of abstract executions, and are therefore Galois connections between the process spaces. 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 upper and lower bound of conservative approximations. However, our use of the upper and lower bound is significantly different, since we apply them in combination (the lower bound for the specification, the upper bound 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. [13] 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 [13], 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 upper and lower bounds 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. In the work presented in [13], this translates in two adjoints per pair of categories.

The study of heterogeneous systems is the central theme of the

Ptolemy project [8]. One of the innovative concepts in the design of the Ptolemy II infrastructure is the notion of *domain polymorphism* [9]. An actor (agent) is domain polymorphic if it can be used indifferently in several models of computation. To check whether an actor can be used in a particular model, 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. We also introduce a similar notion. In our framework, an agent can be used in different models of computation if it has an exact representation in such models. The notion of abstraction in the form of a conservative approximation and its inverse provides us with the appropriate interpretation of an agent from one model in another model. 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.

2. AGENT ALGEBRAS

Our notion of conservative approximation is based on the framework of Agent Algebra [12]. Informally, an agent algebra $\mathcal{Q}$ is composed of a domain D which contains the agents under study for the algebra, and of certain operators that formalize the most common operations of the models of computation used in embedded system design. Different models of computation are constructed by providing different definitions for the domain of agents and the operators. The algebra also includes a master alphabet $\mathcal{A}$ that is used as the universe of "signals" that agents use to communicate with other agents.

Definition 2.1 (Agent Algebra). An *agent algebra* $\mathcal{Q}$ has a domain $\mathcal{Q}.D$ of *agents*, a *master alphabet* $\mathcal{Q}.\mathcal{A}$, and three operators: *renaming*, *projection* and *parallel composition*, denoted by $\mathit{rename}(r)$, $\mathit{proj}(B)$ and $\|$. Each agent $p \in \mathcal{Q}.D$ is associated with an *alphabet* $A \subseteq \mathcal{Q}.\mathcal{A}$.

The operators of the algebra are partial functions on the domain D and have an intuitive correspondence with those of most models of concurrent systems. The operation of renaming, which takes as argument a renaming function r on the alphabet, corresponds to the instantiation of an agent in a system. The renaming function is required to be a bijection, so that renaming is prevented from altering the structure of the agent interface, by for example "connecting" two signals together. Projection corresponds to hiding a set of signals, and takes the set B of signals to be retained as a parameter. Hence it corresponds to an operation of scoping. Finally, parallel composition corresponds to the concurrent "execution" of two agents. It is possible to define other operators. We prefer to work with a limited set and add operators only when they cannot be derived from existing ones. The operators presented here are sufficient for the scope of this work.

The three operators must satisfy certain axioms that formalize their intuitive behavior and provide some general properties that we want to be true regardless of the model of computation. For example, parallel composition must be associative and commutative, and the alphabet of the result must be obtained as the union of the original alphabets, thus ruling out the possibility of a simultaneous projection. Similar requirements on the alphabets of projection and renaming also make sure that these operators effectively performs their respective function. The definition of the operators is otherwise unspecified, and depends on the particular agent model being considered.

To illustrate our framework, we build an example of an agent algebra $\mathcal{Q}^C$ that can be used to model event-based systems in continuous time. In this model, a pair (a, τ) is an event that denotes the occurrence of an action $a \in \mathcal{A}$, at a time $\tau \in \mathbb{R}$. 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. Note that the order of events in a behavior is derived from the order on the time stamps. In addition, an agent is characterized 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. The construction of an agent as a set of behaviors is quite general and can be employed with other models of behavior. In addition, we may sometimes restrict the acceptable sets of behaviors for agents to model requirements such as receptiveness, or to avoid certain phenomena, such as Zeno behaviors. Since the operators must be closed under these restrictions, the resulting algebra forms a subalgebra of a more general, unrestricted model. We will not, however, be concerned with these aspects in this paper.

To complete the description of the model we must define the operators of the algebra. Projection simply consists of removing from the behaviors of an agent those events that correspond to actions that must not be retained. For example, the projection of the behavior x above to the set of signals $\{a, c\}$, is simply

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

Renaming is defined similarly by extending to events and to the sets I and O the application of the renaming function. For instance, if a renaming function r maps signal a to in_1, signal b to in_2 and signal c to out, then

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

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 concurrently an output of the other agent. Thus

$$O = O_1 \cup O_2,$$
$$I = (I_1 \cup I_2) - (O_1 \cup O_2).$$

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,

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

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. It is easy to verify that 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 [7] (see below).

It is easy to construct a similar, but more abstract, agent algebra $\mathcal{Q}^D$ to be used for event-based systems in *discrete* time. It is in fact sufficient to restrict the use of the time stamp τ to the set of integers $\mathbb{Z}$. The remaining definitions are unchanged.

The algebras $\mathcal{Q}^C$ and $\mathcal{Q}^D$ satisfy the requirements of our framework. Models with different structure are also possible. For instance, agents may be as simple as their alphabet (thus ignoring any behavior), or might include complex performance parameters. Similarly, the operators might be defined differently. For example, parallel composition might involve a fixed-point computation. In all cases, only the requirements of the algebra must be satisfied in order for the model to fit in our framework.

Our continuous and discrete time models could also be expressed in the Tagged-Signal Model (TSM) [7]. 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 the form of agent algebras 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 encoded in TSM. The specialization of conservative approximations to this case is part of our future work.

The notion of refinement in each model of computation is represented by adding a preorder (or a partial order) on the agents, denoted by the symbol $\preceq$. The result is called an *ordered agent algebra*. We require that the operators in an ordered agent algebra be monotonic relative to the ordering. However, since these are partial functions, this requires generalizing monotonicity to partial functions. The following definition gives two different generalizations. Later we discuss which of these best suits our needs.

Definition 2.2 ($\top$-$\bot$-monotonic). Let D_1 and D_2 be preordered sets and let f be a partial function from D_1 to D_2. Let

$$\begin{aligned} D_2^{\top} &= D_2 \cup \{\top\}, \\ D_2^{\bot} &= D_2 \cup \{\bot\} \end{aligned}$$

where $\top$ and $\bot$ are new greatest and least elements, i.e.,

$$p_2 \preceq \top \wedge \top \not\preceq p_2$$

and

$$p_2 \not\preceq \bot \wedge \bot \preceq p_2,$$

respectively, for every p_2 in D_2. Let $f^{\top}$ and $f^{\bot}$ be the total functions from D_1 to $D_2^{\top}$ and $D_2^{\bot}$, respectively, such that for all p_1 in D_1

$$f^{\top}(p_1) = \begin{cases} f(p_1), & \text{if } f(p_1) \text{ is defined;} \\ \top, & \text{otherwise;} \end{cases},$$

$$f^{\bot}(p_1) = \begin{cases} f(p_1), & \text{if } f(p_1) \text{ is defined;} \\ \bot, & \text{otherwise.} \end{cases}$$

We say the function f is $\top$-*monotonic* if $f^{\top}$ is monotonic. Analogously, the function f is $\bot$-*monotonic* if $f^{\bot}$ is monotonic.

In this work we always interpret the refinement relation $p \preceq p'$ to mean intuitively that p can be substituted for p' in any context. Let now f be a partial function and assume $f(p)$ is undefined. Then intuitively, $f(p)$ cannot be substituted for any other agent in D_2, except for another undefined expression. This is always the case if f is $\top$-monotonic, since, in that case, $f(p) = \top$ together with $f(p) \preceq f(p')$ imply that $f(p') = \top$, i.e., $f(p')$ is also undefined. Conversely, if f is $\bot$-monotonic and $f(p)$ is undefined, then $f(p)$ can be substituted for any agent in the system. Therefore, only $\top$-monotonic functions are consistent with our interpretation of the order. Hence, we require that the operators of projection, renaming and parallel composition in an ordered agent algebra be $\top$-monotonic relative to the agent order.

It is easy to introduce an order for the algebras $\mathcal{Q}^C$ and $\mathcal{Q}^D$ that makes the operators $\top$-monotonic. If p_1 and p_2 are agents, we say that $p_1 \preceq p_2$ if and only if $I_1 = I_2$, $O_1 = O_2$ and $P_1 \subseteq P_2$. In other words, we require that all the behaviors of the implementation are also behaviors of the specification. This order is similar to the one used in traditional language-containment verification techniques. The reader can easily verify that the operators of $\mathcal{Q}^C$ and $\mathcal{Q}^D$ satisfy the definition of $\top$-monotonicity. Other orders, that for example relax the requirement that p_1 and p_2 have the same inputs and outputs, are also possible but are beyond the scope of this paper [12].

The notion of $\top$-monotonicity has profound implications on the compositionality rules in our framework. A compositional verification strategy consists of partitioning a particular verification problem into a number of smaller problems that are collectively easier to solve. For refinement verification, this can often be accomplished by breaking up a specification and an implementation in terms of their components, and by considering refinement relationships between the corresponding components. The original verification problem can be solved in this way when the parallel operator is monotonic. For a generic partial operator f, $\top$-monotonicity is related to the traditional notion of monotonicity by the following result.

Theorem 2.3. Let f be a $\top$-monotonic partial function. If $p \preceq p'$ and $f(p')$ is defined, then $f(p)$ is defined and $f(p) \preceq f(p')$.

In particular, if the parallel composition operator $\|$ of an agent algebra is $\top$-monotonic relative to the refinement order, the above result reduces to the following.

Corollary 2.4. If $p_1 \preceq p_1'$, $p_2 \preceq p_2'$ and $p_1' \parallel p_2'$ is defined, then $p_1 \parallel p_2$ is defined and $p_1 \parallel p_2 \preceq p_1' \parallel p_2'$.

Henzinger et al. [4] propose to distinguish between *interface* and *component* algebras. The above result shows that because parallel composition is $\top$-monotonic in an ordered agent algebra, it supports an inference rule identical to the "compositional design" rule for interface algebras. Conversely, component algebras have a "compositional verification" rule that corresponds to $\bot$-monotonic functions. This suggests that the ordering of a component algebra cannot be interpreted as indicating substitutability. Interface and component algebras were introduce to distinguish between agents that make assumptions relative to their environment (the interfaces), and agents that do not (the components). These notions can be handled in our framework by using appropriate models (for example, including *failure* behaviors [5]), that always employ $\top$-monotonic operators. Thus, we distinguish between interfaces and components in our framework based on their level of abstraction, rather than on the inference rule they support.

3. CONSERVATIVE APPROXIMATIONS

As discussed in the introduction we relate different agent algebras by means of conservative approximations. A conservative ap-

proximation from $\mathcal{Q}$ to $\mathcal{Q}'$ is a pair $\Psi = (\Psi_l, \Psi_u)$, where Ψ_l and Ψ_u are functions from $\mathcal{Q}.D$ to $\mathcal{Q}'.D$. The first mapping is an upper bound of the agent relative to the order of the algebra: for instance, the abstract agent represents all of the possible behaviors of the agent in the more detailed domain, plus possibly some more. The second is a lower bound: the abstract agent represents only possible behaviors of the more detailed one, but possibly not all.

We define conservative approximations as abstractions that maintain a precise relationship between the orders in the two agent algebras.

Definition 3.1 (Conservative Approximation). Let $\mathcal{Q}$ and $\mathcal{Q}'$ be ordered agent algebras, and let Ψ_l and Ψ_u be functions from $\mathcal{Q}.D$ to $\mathcal{Q}'.D$. We say that $\Psi = (\Psi_l, \Psi_u)$ is a *conservative approximation from $\mathcal{Q}$ to $\mathcal{Q}'$* if and only if for all agents p and q in $\mathcal{Q}.D$,

$$\Psi_u(p) \preceq \Psi_l(q) \Rightarrow p \preceq q.$$

Thus, when used in combination, the two mappings allow us to relate refinement verification results in the abstract domain to results in the more detailed domain. Hence, the verification can be done in $\mathcal{Q}'$, where it is presumably more efficient than in $\mathcal{Q}$. The conservative approximation guarantees that this will not lead to a false positive result, although false negatives are possible depending on how the approximation is chosen.

It is easy to show that 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.

A simple example of a conservative approximation can be constructed as follows. Consider the agent algebra $\mathcal{Q}^{\mathbb{R}^{\preceq}}$ where each agent is simply a real number (representing, for example, maximum power dissipation). We order $\mathcal{Q}^{\mathbb{R}^{\preceq}}$ by defining $p \preceq p'$ if and only if p is less than or equal to p'. Let $\mathcal{Q}^{\mathbb{N}}$ be the analogous ordered agent algebra where each agent is a non-negative integer, rather than a real number. Then, $\Psi = (\Psi_l, \Psi_u)$ is a conservative approximation from $\mathcal{Q}^{\mathbb{R}^{\preceq}}$ to $\mathcal{Q}^{\mathbb{N}}$, where

$$\begin{aligned} \Psi_l(p) &= \lfloor p \rfloor \\ \Psi_u(p) &= \lceil p \rceil \end{aligned}$$

(i.e., the floor and the ceiling, respectively, of the real number p). This example is typical: neither the floor function, nor the ceiling function, when used alone, would satisfy the requirements of a conservative approximation.

An example of a conservative approximation from our continuous time model $\mathcal{Q}^C$ to the discrete time model $\mathcal{Q}^D$ can be built by first considering a function h between the behaviors of $\mathcal{Q}^C$ and $\mathcal{Q}^D$. Specifically, h could be the function that truncates the time stamp of each continuous time event to its nearest integer. For example,

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

A conservative approximation on the agents is essentially a pair of functions that operate on sets of behaviors. Given a function h on behaviors, there are different ways one could derive a function on sets of behaviors. Here we use the notion of an axiality [12], which is similar to the pre and post images of a binary relation proposed by Loiseaux et al. [10], and to the optimistic and pessimistic process abstractions of Negulescu [11]. This construction can be applied to most behavior-based models, and greatly simplifies the process of creating a conservative approximation between two agent models [12]. For an agent $p = (I, O, P)$, the conservative approximation is of the form

$$\begin{aligned} \Psi_u^C(p) &= (I, O, h(P)) \\ \Psi_l^C(p) &= (I, O, h(P) - h(\mathcal{B}(A) - P)). \end{aligned}$$

Note that Ψ_u^C is effectively an upper bound since, in that case, every behavior of a continuous time agent is represented by an abstract behavior of the discrete time agent. Conversely, the lower bound of p includes an abstract behavior only if p contains all its possible concretizations. For example, $\{(a, 1)\}$ is in $\Psi_l^C(p)$ only if p contains the behavior that has an event (a, τ) for every $\tau \in [1, 2)$. Note also that the approximation loses some of the ordering constraints. For example, the continuous time events $(a, 2.1)$ and $(b, 2.2)$ correspond to events $(a, 2)$ and $(b, 2)$ in the discrete time model. Thus the relative ordering of actions a and b is lost.

Other functions 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 conservative approximations. These, in turn, can be used to represent different implementation strategies in a design flow. This discussion is, however, beyond the scope of this paper.

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 $\mathcal{Q}.D$ that is represented exactly by p' rather than just being bounded by p'. If no agent in $\mathcal{Q}.D$ can be represented exactly, then Ψ is abstracting away too much information to be of much use for verification. If every agent in $\mathcal{Q}.D$ 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 this context, we define the inverse Ψ_{inv} of the conservative approximation Ψ. Normal notions of the inverse of a function are not adequate for this purpose, since Ψ is a pair of functions. We handle this by only considering those agents p for which $\Psi_l(p)$ and $\Psi_u(p)$ have the same value, call it p'. Intuitively, p' represents p exactly in this case, and we therefore define $\Psi_{inv}(p') = p$. 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 3.2 (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 if and only if there exists $p \in \mathcal{Q}.D$ such that $\Psi_l(p) = \Psi_u(p) = p'$. When defined,

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

If the algebra $\mathcal{Q}$ is partially ordered (as opposed to preordered), the inverse of the conservative approximation is uniquely determined and, when defined, the inverse $\Psi_{inv}(p')$ of p' is equal to the unique agent p such that $\Psi_l(p) = \Psi_u(p) = p'$. Otherwise, a choice may be possible among order equivalent agents. In all cases, however, 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 it

preserves the ordering of the agents in both directions if either Ψ_l or Ψ_u is also monotonic. To simplify the presentation, in the rest of this paper we will only be concerned with partially ordered agent algebras.

The inverse of a conservative approximation can be used to formally understand the roles of the upper and the lower bounds. Assume, in fact, that for an agent p, $\Psi_{inv}(\Psi_l(p))$ and $\Psi_{inv}(\Psi_u(p))$ are both defined, It is easy to show that, in this case,

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

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 $p' \in \mathcal{Q}'.D$ determines two equivalence classes in $\mathcal{Q}.D$: the class of the agents p such that $\Psi_u(p) = p'$, and the class of the agents p such that $\Psi_l(p) = p'$. The inverse $\Psi_{inv}(p')$ of p' is defined if and only the greatest element of the first class is equal to the least element of the second class, i.e.,

$$\Psi_{inv}(p') = \max\{p_1 : \Psi_u(p_1) = p'\} \quad (1)$$
$$= \min\{p_1 : \Psi_l(p_1) = p'\}. \quad (2)$$

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

It is easy to determine the inverse Ψ^C_{inv} of the conservative approximation $\Psi^C = (\Psi^C_l, \Psi^C_u)$ from $\mathcal{Q}^C$ to $\mathcal{Q}^D$. In fact, let $p' \in \mathcal{Q}^D.D$ be an arbitrary discrete time agent. For every event (a, τ') of p', there exists a continuous time agent $p \in \mathcal{Q}^C.D$ that nondeterministically executes the same action a with time stamps τ such that $\tau' \leq \tau \leq \tau' + 1$. Thus the inverse is always defined. Note however that, because of the resulting non-determinism, the information on the exact location of the actions within a unit of time in the corresponding continuous time agent is unknown.

3.1 Compositional 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 algebra. 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

$$\mathit{proj}(A)(p_1 \parallel p_2) \preceq p,$$

where p_1, p_2 and p are agents in $\mathcal{Q}.D$. 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 p. We say that a conservative approximation Ψ is a *compositional* conservative approximation if showing

$$\mathit{proj}(A)(\Psi_u(p_1) \parallel \Psi_u(p_2)) \preceq \Psi_l(p)$$

is sufficient to show that the original implementation satisfies its specification. The following definition makes this notion precise.

Definition 3.3 (Compositional Conservative Approx.). 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 $p_1 \in \mathcal{Q}.D$,

$$E[p/\Psi_u(p)] \preceq \Psi_l(p_1) \Rightarrow E \preceq p_1,$$

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

The following theorem provides sufficient conditions that the upper bound of a conservative approximation must satisfy in order to also be compositional.

Theorem 3.4. 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 $\mathcal{Q}.D$, 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 $\mathit{proj}(B)(\Psi_u(p))$ is defined, then

$$\Psi_u(\mathit{proj}(B)(p)) \preceq \mathit{proj}(B)(\Psi_u(p)).$$

S3. If $\mathit{rename}(r)(\Psi_u(p))$ is defined, then

$$\Psi_u(\mathit{rename}(r)(p)) \preceq \mathit{rename}(r)(\Psi_u(p)).$$

The conservative approximation Ψ^C from $\mathcal{Q}^C$ to $\mathcal{Q}^D$ is compositional. There is in fact a general result that guarantees, by applying theorem 3.4, that the axialities of a homomorphism on behaviors (such as our h above) produce a compositional conservative approximation [1, 12]. This is true regardless of the behavior model, as long as it satisfies certain basic properties related to the requirements of our agent algebras. 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 general property

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

for any function f.

4. ABSTRACT INTERPRETATIONS

In section 1.1 we have argued that there exists a close relationship between conservative approximations and abstract interpretations. In this section we explore this relationship in details. We begin by defining Galois connections [3], and by presenting some basic results about them. Later we show how a pair of Galois connections can be used to form a conservative approximation. We then show how to use an abstract interpretation and an additional Galois connection to form a *compositional* conservative approximation.

Definition 4.1 (Galois Connection). Let D and D' be partially ordered sets of agents. A Galois connection $\langle \alpha, \gamma \rangle$ from D to D' consists of an abstraction map $\alpha : D \mapsto D'$ and a concretization map $\gamma : D' \mapsto D$ such that for all $p \in D$ and $p' \in D'$,

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

Galois connections enjoy several properties similar to the ones discussed for conservative approximations. In particular, if $\langle \alpha, \gamma \rangle$ is a Galois connection, then α and γ are monotonic and for all $p \in D$ and $p' \in D'$, $p \preceq \gamma(\alpha(p))$ and $\alpha(\gamma(p')) \preceq p'$ ([10]). In addition, the composition of Galois connections is again a Galois connection.

The relationship between Galois connections and conservative approximations can be understood by introducing a second Galois connection in the reverse direction, i.e., from D' to D. For our notation, we will use symbols $\langle \alpha_u, \gamma_u \rangle$ for a Galois connection from $\mathcal{Q}.D$ to $\mathcal{Q}'.D$, and $\langle \gamma_l, \alpha_l \rangle$ for a Galois connection from $\mathcal{Q}'.D$ to $\mathcal{Q}.D$. Note that in the case of the Galois connection from $\mathcal{Q}'.D$ to

$\mathcal{Q}.D$ 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 4.2. (α_l, α_u) is a conservative approximation from $\mathcal{Q}$ to $\mathcal{Q}'$ if and only if for all agents $p' \in \mathcal{Q}'.D$, $\gamma_u(p') \preceq \gamma_l(p')$.

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 $\langle \gamma_l, \alpha_l \rangle$. Our second 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 $p' \in \mathcal{Q}'.D$ if and only if γ_u and γ_l are equal, and are "mutually" injective.

Theorem 4.3. For all agents $p' \in \mathcal{Q}'.D$, $\Psi_{inv}(p')$ is defined and is equal to p if and only if

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

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. When that is the case, $\langle \Psi_u, \Psi_{inv} \rangle$ 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 $\mathcal{Q}.D$. Note that 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 connections. This occurs when the equivalence classes induced by Ψ_u and Ψ_l do not have the necessary greatest and lowest element, as described in equation 1.

Galois connections can be used to build abstract interpretations. Abstract interpretations were originally developed for static analysis of sequential programs in optimizing compilers [2]. They have also been used for abstracting and formally verifying models of both sequential and reactive systems. Abstract interpretations between agent algebras can be defined as follows.

Definition 4.4 (Abstract Interpretation). Let $\mathcal{Q}$ and $\mathcal{Q}'$ be partially ordered agent algebras. Then $\mathcal{Q}'$ is an abstract interpretation of $\mathcal{Q}$ if and only if there exists a Galois connection $\langle \alpha, \gamma \rangle$ from $\mathcal{Q}.D$ to $\mathcal{Q}'.D$ such that for all p'_1 and p'_2 in $\mathcal{Q}'.D$,

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 $\mathit{proj}(B)(p'_1)$ is defined, then
$$\alpha(\mathit{proj}(B)(\gamma(p'_1))) \preceq \mathit{proj}(B)(p'_1),$$
3. if $\mathit{rename}(r)(p'_2)$ is defined, then
$$\alpha(\mathit{rename}(r)(\gamma(p'_2))) \preceq \mathit{rename}(r)(p'_2).$$

The agent algebra $\mathcal{Q}^D$ is an abstract interpretation of the algebra $\mathcal{Q}^C$. In fact, the upper bound Ψ_u^C and the inverse Ψ_{inv}^C of the conservative approximation from $\mathcal{Q}^C$ to $\mathcal{Q}^D$ form a Galois connection (because they are derived as an axiality) which satisfies the conditions of definition 4.4.

A fundamental result of our theory shows that the three conditions of definition 4.4 are equivalent to the conditions S1 through S3 of theorem 3.4. Therefore, abstract interpretations, when used in conjunction with a second Galois connection from $\mathcal{Q}'.D$ to $\mathcal{Q}.D$ induce *compositional* conservative approximations, as described in theorem 4.2. This result relies on $\top$-monotonicity of the operators of the algebra.

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. 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. Abstract interpretations, however, are unable to guarantee that a positive refinement verification result at the abstract level implies a positive refinement verification result at the concrete level. In other words, if $\alpha(p_1) \preceq \alpha(p_2)$, then $p_1 \preceq p_2$ does not necessarily hold.

Conservative approximations, on the other hand, employ two mappings to guarantee the above verification result. It is easy to show the differences between conservative approximations and abstract interpretations by using our continuous and discrete time models. 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 obviously 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 abstract interpretation makes use of only the upper bound Ψ_u^C of the conservative approximation. The behavior x above, for example, is represented in $\Psi_u^C(q)$ as

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

Assume now $p \in \mathcal{Q}^C.D$ is such that $\Psi_u^C(p) \preceq \Psi_u^C(q)$. Given our definitions, x' could be a behavior of $\Psi_u^C(p)$. Note however that x' in $\Psi_u^C(p)$ could be obtained as the abstraction of a different behavior, such as

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

which clearly violates our specification. In other words, $p \not\preceq q$.

The above verification technique based on abstract interpretations is unsound because q, our specification, is not represented exactly in $\mathcal{Q}^D$. For example, x' is not contained in $\Psi_l^C(q)$. If it were, q would have to contain behaviors, such as y, where action b *precedes* action a in the time span from 3 to 4, which contradicts the specification. Hence $\Psi_u^C(q) \neq \Psi_l^C(q)$. Conservative approximations avoid this problem by explicitly using the lower bound. Note, in fact, that $\Psi_l^C(q)$ is 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 $\Psi_u^C(p) \preceq \Psi_l^C(q)$ (note the use of the lower bound for the specification), 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 abstract interpretations. 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. For this case, abstract interpretations 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.

5. REFINEMENT AND POLYMORPHISM

In section 3 we have characterized an 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 following 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 connection 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).

Suppose now that $\mathcal{Q}$ and $\mathcal{Q}'$ are agent algebras, and that $\Psi = (\alpha_l, \alpha_u)$ is a conservative approximation from $\mathcal{Q}$ to $\mathcal{Q}'$ induced by

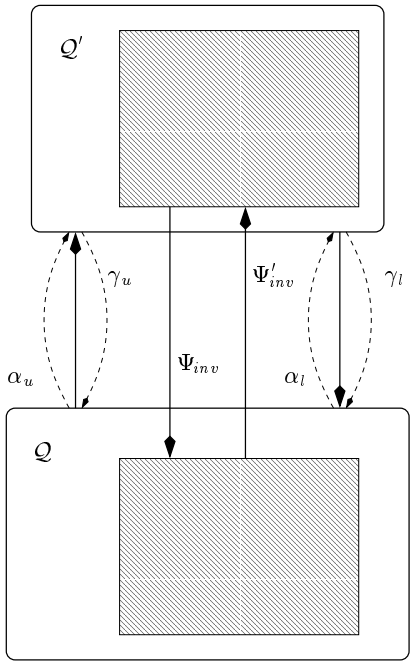

Figure 1: Abstraction, Refinement and their inverses

a pair of Galois connections $\langle \alpha_u, \gamma_u \rangle$ and $\langle \gamma_l, \alpha_l \rangle$. Theorem 4.2 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). Note also that 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 figure 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 bound do not coincide). We say that an agent p that can be represented exactly in two different models of computation $\mathcal{Q}$ and $\mathcal{Q}'$ is *polymorphic relative to $\mathcal{Q}$ and $\mathcal{Q}'$*. Note that our 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. This is unlike other notions of polymorphism that rely upon a common underlying semantics, such as the automata model in Ptolemy II [9].

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. 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 in the opposite direction 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 closure or interior systems. 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. Equation 1 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 5.1. Let $\mathcal{Q}$ and $\mathcal{Q}'$ be partially ordered agent algebras and 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

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

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_l, 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 makes 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.

6. CONCLUSIONS

In this paper we have used the framework of Agent Algebra to present and discuss abstraction and refinement operations in the form of conservative approximations. We have compared conservative approximations to abstract interpretations and we have shown 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 for heterogeneous systems that employ several models of computation. In addition, we have used the inverse of a conservative approximation to identify components that can be used indifferently in several models, thus enabling reuse across domains of computation.

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 of the algebra in order to correctly translate expressions. Note that in this case the homomorphism becomes similar to a functor between categories, where a category has behaviors as objects and the operators of the algebra as morphisms.

An agent algebra that uses behaviors as its underlying model 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, in this case, may not necessarily be defined if the resulting agent does not 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 algebra, and while still functioning as the bound of a conservative approximation. This would constitute a generalization of the techinique proposed by Loiseaux et al. [10] on property-preserving abstractions in the context of transition systems.

7. REFERENCES

[1] Jerry R. Burch. *Trace Algebra for Automatic Verification of Real-Time Concurrent Systems*. PhD thesis, School of Computer Science, Carnegie Mellon University, August 1992.

[2] P. Cousot and R. Cousot. 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*, pages 238–252, Los Angeles, California, 1977. ACM Press, New York, NY.

[3] P. Cousot and R. Cousot. Comparing the Galois connection and widening/narrowing approaches to abstract interpretation, invited paper. In M. Bruynooghe and M. Wirsing, editors, *Proceedings of the International Workshop Programming Language Implementation and Logic Programming, PLILP '92,*, Leuven, Belgium, 13–17 August 1992, Lecture Notes in Computer Science 631, pages 269–295. Springer-Verlag, Berlin, Germany, 1992.

[4] Luca de Alfaro and Thomas A. Henzinger. Interface theories for component-based design. In Henzinger and Kirsch [6], pages 148–165.

[5] David L. Dill. *Trace Theory for Automatic Hierarchical Verification of Speed-Independent Circuits*. ACM Distinguished Dissertations. MIT Press, 1989.

[6] Thomas A. Henzinger and Christoph M. Kirsch, editors. *Embedded Software*, volume 2211 of *Lecture Notes in Computer Science*. Springer-Verlag, 2001.

[7] E. A. Lee and A. L. Sangiovanni-Vincentelli. A framework for comparing models of computation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, 17(12):1217–1229, December 1998.

[8] Edward A. Lee. Overview of the Ptolemy project. Technical Memorandum UCB/ERL M03/25, University of California, Berkeley, July 2003.

[9] Edward A. Lee and Yuhong Xiong. System-level types for component-based design. In Henzinger and Kirsch [6].

[10] Claire Loiseaux, Susanne Graf, Joseph Sifakis, Ahmed Bouajjani, and Saddek Bensalem. Property preserving abstractions for the verification of concurrent systems. *Formal Methods in System Design*, 6:1–35, 1995.

[11] Radu Negulescu. *Process Spaces and the Formal Verification of Asynchronous Circuits*. PhD thesis, University of Waterloo, Canada, 1998.

[12] Roberto Passerone. *Semantic Foundations for Heterogeneous Systems*. PhD thesis, Department of EECS, University of California at Berkeley, May 2004.

[13] Vladimiro Sassone, Mogens Nielsen, and Glynn Winskel. Models for concurrency: Towards a classification. *Theoretical Computer Science*, 170:297–348, 1996.