Contract-based compositional analysis for reactive systems in RTEdge™, an AADL-based language

Technical Report 2013-607

Ernesto Posse

School of Computing
Queen's University
Kingston, Ontario, Canada

September 2013

Abstract

This report investigates compositional reasoning techniques for RTEdge™ [Ghe11, Pos13a], a language for modelling real-time embedded systems based on AADL [SAE12] and developed by Edgewater Computer Systems Inc. The RTEdge™ platform supports formal verification of software models as described in [Ghe11], but it performs a monolithic analysis, by model-checking the entire model at once. To avoid or alleviate the state explosion problem, we propose the use of *assume/guarantee contract-based reasoning*, a family of compositional analysis semi-automatic techniques which leverage the structure of the model under consideration. In particular, we adapt the generic theoretical framework for assume/guarantee contracts proposed in [BDH+12a] to the RTEdge™ setting where contracts would be specified using the Property Specification Language (PSL) [IEE05] an IEEE standard.

In this report we describe: 1) how PSL could be adapted to specify contracts for RTEdge™ models, 2) how to establish the consistency and conformance of specifications and contracts between different language elements such as protocols, interfaces and capsules, 3) how the framework from [BDH+12a] can be used with PSL to perform compositional analysis, including *quotienting*, *i.e.*, finding a specification that can "complete" a model and 4) a description of the tool support required to implement this framework with a quick survey of available tools that could be used.

Contents

List of Algorithms

List of Figures

List of Tables

1 Introduction

The RTEdge™ platform, developed by Edgewater Computer Systems Inc., is a model-driven software development environment for real-time embedded systems (RTE), which supports the specification, design, analysis, code generation, and testing of software for RTE systems. RTEdge™'s analysis capabilities include causality (flow) analysis, schedulability analysis, virtual-time simulation and formal verification. At the core of the development platform is the RTEdge™ modelling language [Ghe11, Pos13a], a language based on the Architecture Analysis & Design Language (AADL) [SAE12] an industry standard used in a variety of sectors, including military, government and commercial enterprise. RTEdge™'s target application domains include areas such as automotive, avionics, and telecommunications systems.

The design and implementation of RTE systems is notoriously difficult. Formal analysis is necessary to ensure that the software satisfies its requirements and to try to eliminate errors early on in the design stage of the development process. To this end, the RTEdge™ platform supports formal verification by means of model-checking [CGP99, BK08]. This capability, described in [Ghe11], consists on translating the entire model into Promela, the input language for the well-known Spin model-checker [Hol04]. While Spin is widely regarded as an efficient model-checker, this monolithic approach does not scale well. The addition of new components to a model leads to an exponential increase in the number of states in the system, a problem commonly known as the "state explosion problem". The model-checking literature provides numerous techniques to cope with this problem (see *e.g.*, [CGP99]). One of the main approaches is *compositional reasoning* which relies on leveraging the structure of the model under consideration. Compositional reasoning is essentially a "divide-and-conquer" approach as it reduces the problem of analyzing a model to the problem of analyzing its parts.

Compositional analysis has several advantages in addition to coping with the state explosion problem. One of main advantages is that it reduces the necessity of repeating the analysis whenever a change is made to the design model. If we have a compositional analysis capability, then replacing a component with another implementation that satisfies the same properties of interest, will preserve the properties and correctness of the composite model, making unnecessary to re-analyze the whole model. Furthermore, compositional reasoning provides the basis for *incremental analysis*. If we replace a component with another implementation that satisfies other properties, or we add a new component, we may need to analyze the modified or new components, but we will not be required to repeat the analysis for unaffected components. These characteristics make compositionality a highly desirable feature in any analysis framework, and have an obvious impact on the scalability of formal analysis.

There has been successful research on compositional reasoning for programming languages and hardware systems (*e.g.*, [BCC97, CLM89, CGP99, GL94, LG98, PV02, Cha94]) but there has been little attention paid to modelling languages in general and for RTE systems in particular.

Compositional techniques have been studied for a wide variety of formalisms and denotational and operational models such as Kahn data-flow process networks [LS89], I/O-automata [LT87, Jon90], timed I/O-automata [KL04, BO01], probabilistic I/O-automata [WSS94], hybrid I/O automata [LSV03], interface automata [dAH01], I/O labelled transition systems [CCJK12], various process calculi such as CCS [DG99, RR01] and CSP [WW09], etc. However it is not clear if, and how, these frameworks could be directly applicable to RTEdge™, since each of these targets particular operational models and/or specification formalisms, making assumptions which may not be satisfied by RTEdge™. In order to use any such framework, we are forced to map RTEdge™ concepts, specifications and models into the concepts, specifications and models required by the particular framework. This task is not trivial, usually requiring a translation from RTEdge™ to the formalism in question, assuming that all relevant concepts can be mapped, which is far from clear.

Compositional techniques have also been used to reason about different kinds of properties and to perform different kinds of analysis such as performance analysis [Sta00], schedulability analysis [ELSS07, Shi06, SS98] and security [LA10]. However, the focus of this project is on the satisfaction of functional requirements, and in particular about the ordering of sequences of events and interactions between the components of a model.

A particular kind of compositional analysis techniques, known as *contract-based* or *assume/guarantee (A/G) reasoning* [Pnu84, CGP99, GL94, KV97], provide a useful approach to tackle the state explosion problem while supporting and even encouraging component-based design [Gie00]. In this paradigm, each component is annotated with a *contract* consisting of a set of *assumptions* about how the environment of the component is expected to behave, and a set of *guarantees* specifying the behaviour of the component if the

assumptions hold. The heart of the analysis is to establish that a composite component satisfies its contract if its sub-components satisfy their respective contracts. There is, however, an inherent difficulty in trying to apply this technique: if a composite component K contains two parts K_1 and K_2, and we wish to establish that K satisfies a property φ, given that K_1 satisfies some property φ_1 and K_2 satisfies some property φ_2, we need a "compositional proof rule" which allows us to infer that K satisfies a property φ from the knowledge that each K_i satisfies φ_i. But this leads us to a circular argument, since K_1 and K_2 may interact, to establish that K_1 satisfies φ_1, we may need to assume that K_2 satisfies φ_2, and to establish that K_2 satisfies φ_2, we may need to assume that K_1 satisfies φ_1!

Much of the research on the assume-guarantee paradigm has been about how to resolve this circularity and to establish under what conditions we can apply a compositional proof rule to infer the satisfaction of a property by a composite component given the satisfaction of properties by the sub-components. Different theoretical frameworks have been proposed to support just such reasoning rule and resolve the circularity, but most of them prove the correctness of such compositional rule for specific formalisms or operational models, thus relying on specific assumptions about the kinds of components modelled. This entails, for example, specific requirements or restrictions on the kind of specifications under consideration (*e.g.*, timed I/O automata, interface automata, sets of traces, etc.), the kind of component composition allowed (*e.g.*, synchronous product of automata, trace intersection), the meaning of a component satisfying a contract (*e.g.*, satisfaction as refinement), or the notion of refinement between components (*e.g.*, trace inclusion, simulation preorders, game semantics).

However, recent work described in [BDH+12a] provides a theoretical framework for assume/guarantee reasoning which supports a sound compositional proof rule and is independent of the specific formalisms used to describe components, assumptions and guarantees, as long as the specification theories used satisfy certain core requirements. In this report we show how we can use this theoretical framework to do assume/guarantee reasoning for RTEdge™ models where contracts are specified using the Property Specification Language (PSL) [IEE12a, IEE05, Acc04], a temporal logic which combines Sequential Extended Regular Expressions (SEREs), Linear Temporal Logic (LTL) and (optionally) Computation Tree Logic (CTL). PSL is an IEEE standard with an increasing user-base among designers of embedded systems, hardware and software.

The framework from [BDH+12a] has several advantages: 1) it is applicable to a wide range of specification formalisms and models, and in particular it is applicable to RTEdge™ and PSL; 2) it yields an intuitive solution to compositional analysis; and 3) as we will show, when instantiated to logical theories such as PSL, it suggests a straight-forward approach to automatizing the compositional verification problem.

Report organization The rest of this report is organized as follows: in Subsection 1.1 we provide a working example as a motivation; in Subsection 1.2 we describe the problems that we are trying to solve in more detail and the scenarios in which these arise; in Section 2 we describe the work-flow implied by this framework; in Section 3 we describe the theoretical framework from [BDH+12a]; in Section 4 we discuss the kind of specifications and contracts that we will allow associated to different elements of the RTEdge™ language; in Section 5, how to establish the conformance or compliance between specifications and contracts; in Section 6 we describe the compositional analysis itself: in Subsection 6.2 we address the issue of verifying atomic capsules; in Subsection 6.3 we apply the theoretical framework to the basic problem of compositional reasoning; in Section 7 we show how to apply compositional analysis to do incremental analysis; in Section 8 we deal with what we call the "missing part problem" ; in Section 9 we discuss how to perform the checks required by the framework as presented in the previous sections, including the necessary tool support; and in Section 10 we conclude with a summary of the proposed framework and some recommendations for its implementation.

1.1 A working example

In this section we describe an RTEdge™ model that we will use as a working example throughout this report. A brief description of the RTEdge™ language, its syntax and informal semantics can be found in Appendix A. A brief description of the core elements of PSL that we will use can be found in Appendix B.

The model presented here is a toy model, but one that describes a composite structure with complex patterns of interaction, not unlike those found on real systems. Furthermore, the use of a small model to discuss the analysis framework is useful, as a complicated model might obscure the ideas being presented.

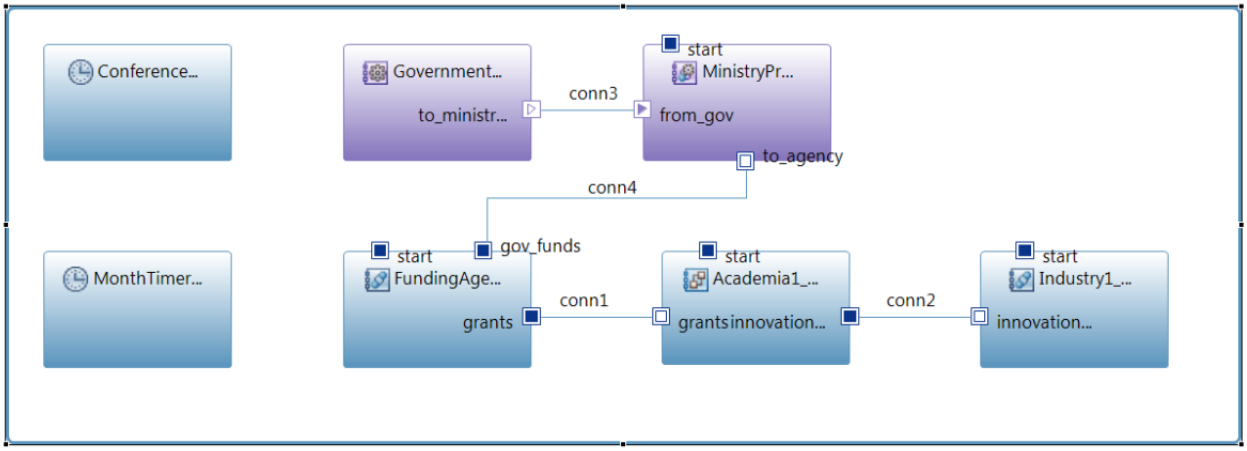

Figure 1: Academics application. Top-level view.

Protocol name	Input signals	Output signals
CallForPapers	paper	call, accept, reject, revise
Funds	amount	–
Publication	–	paper
ResearchGrant	grant_application, publication	funds, reject
ResearchInnovation	request_paper	paper
Transaction	insert_coin, coffee_button, tea_button	more, change, coffee, tea

Table 1: Protocols of the Academia application. The types associated to signals are omitted.

The model describes academics applying for funding to a funding agency, writing papers, submitting papers to a conference, and interacting with a vending machine. Hence the model includes several components representing academics, funding agencies, conferences, vending machines, paper repositories and industries reading papers. Since the current version of RTEdge^{TM} does not support replicated capsules, we are limited to one entity of each kind.

Top-level application

Figure 1 on page 3 shows the top-level view of the application. It consists of two timers (ConferenceTimer, MonthTimer), an external task capsule (Government), a proxy capsule (MinistryProxy), two atomic capsules (FundingAgency1, Industry1) and a composite capsule (Academia1). The Academia1 composite capsule is shown in Figure 5 on page 6. The protocols used by this application are summarized in Table 1 on page 3.

Timers, external task capsules and proxies

The RTEdge^{TM} platform supports a variety of time units, with the coarsest being the second. In this application we will use the convention that a second represents one month. The ConferenceTimer issues a timeout signal every twelve months (*i.e.*, seconds) while the MonthlyTimer issues a timeout signal every month (*i.e.*, every second).

The Government external task capsule has a unique independent system input called funds which represents funds being allocated by the government to do research. This external task capsule has only one OS port connected to the proxy MinistryProxy, representing the ministry responsible for allocating research funds to research funding agencies. Its behaviour is shown in Figure 2 on page 4. Essentially it acts as a transducer, receiving funds from the Government and passing them to the FundingAgency1, using the protocol Funds (see Table 1 on page 3).

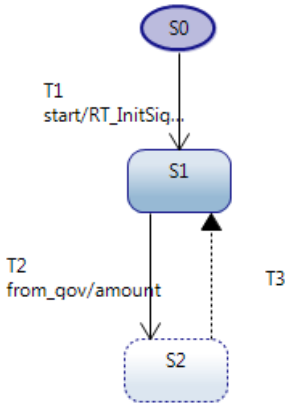

Figure 2: MinistryProxy.

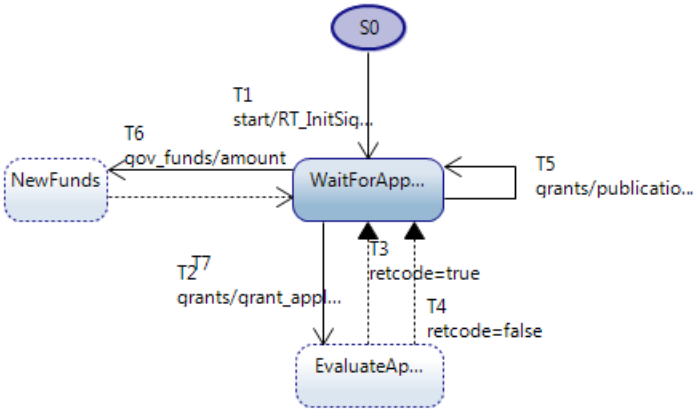

Figure 3: The Funding Agency capsule.

The Funding Agency capsule

The FundingAgency1 capsule has a start port, as do all other normal atomic capsules. This port is used to trigger the initial transition in the capsule's state machine. Additionally it has a gov_funds port where it receives amount signals from the MinistryProxy, and a grants port where it interacts with the academic, by accepting grant_application and publication signals, and sending funds or reject signals. The protocol of the grants port is ResearchGrant (see Table 1 on page 3).

Figure 3 on page 4 shows the behaviour of the FundingAgency1. In the first stable state WaitForApp it listens to grant_application and publication signals on the grants port, and to gov_funds signals on the gov_funds port.

When a gov_funds signal arrives, it goes to the transient state NewFunds where it adds the funds to an internal data attribute storing the available funds, and returns to the WaitForApp state.

When a grant_application signal arrives, it goes to the EvaluateApp state where it executes an internal activity and decides whether to grant the funds or reject the application. Depending on the result of this action, it will either send the funds signal (with an amount) or the reject. In both cases it returns to the WaitForApp state.

The publication signals are ignored.

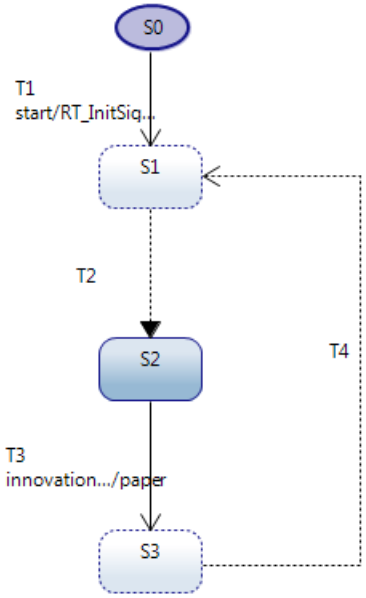

Figure 4: The Industry capsule.

The Industry capsule

In this model the role of the industry capsule is simply to obtain published papers, *i.e.*, papers which have been accepted by the conference and sent to the repository of papers. The Industry1 capsule has a unique port innovations with protocol ResearchInnovation (see Table 1 on page 3).

The Industry1 capsule's behaviour is shown in Figure 4 on page 5. Essentially it is a simple loop where the capsule requests a paper from the repository, waits for the repository to send a paper, and repeats this forever.

The Academia capsule

The Academia1 composite capsule is shown in Figure 5 on page 6. It consists of four sub-capsules: Academic1, Conference1, VendingMachine1, and Repository1. It has a port grants which relays messages to and from the Academic1 capsule, and a port innovations which relays messages to and from the paper repository.

The Repository capsule The Repository1 capsule acts as a store for papers accepted for publication by the Conference1 capsule, and accepts requests for papers from the Industry1 capsule. It has two ports: publish with protocol Publication, and read with protocol ResearchInnovation (see Table 1 on page 3).

The Repository1 capsule's behaviour is shown in Figure 6 on page 6. The capsule maintains an array of papers as a data attribute. In the NoNewPapers state, there are either no papers in the array, or the Industry1 capsule has read all available papers. In this state it can accept a new paper on the publish port and then settle in the NewPaper state where it can accept more papers as well as paper requests on the read. On the NoNewPapers the read port is deferred so that if a request for a paper arrives when no new papers are available, the request will be queued up until it can be consumed in the NewPaper state.

The Conference capsule The Conference1 capsule has two ports: cfp with protocol CallForPapers and papers with protocol Pubication (see Table 1 on page 3). The former is used to interact with the Academic1 capsule, and the later is used to send papers to the Repository1 capsule.

The Conference1 capsule's behaviour is shown in Figure 7 on page 7. The resting state of the capsule is the Idle state, where it lets months pass (by accepting signals from the MonthTimer and once a timeout

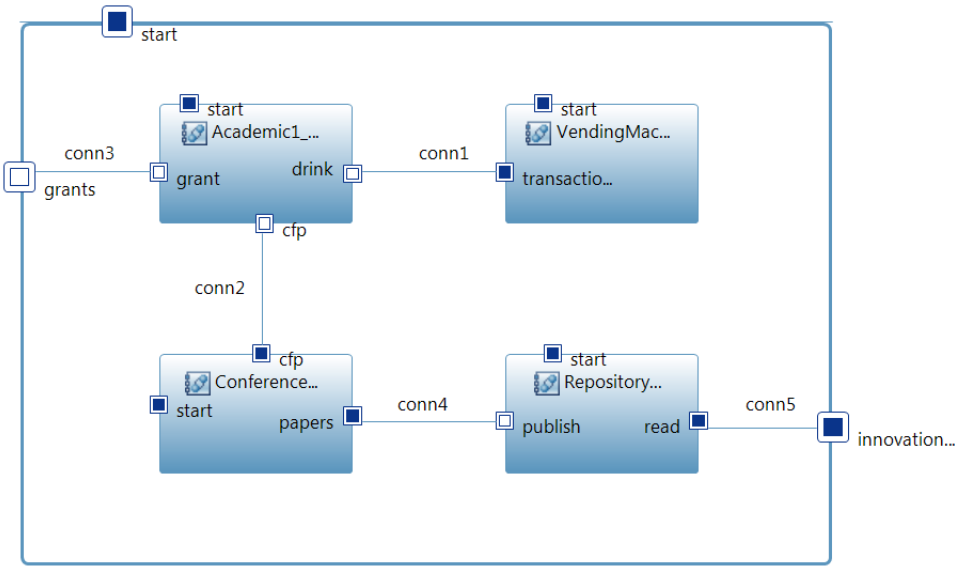

Figure 5: Academia composite capsule.

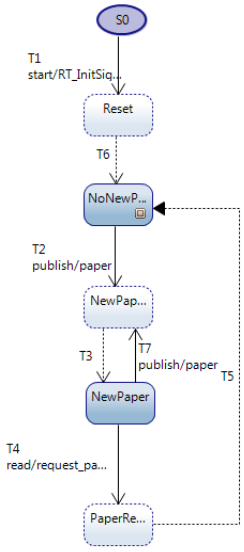

Figure 6: The Repository capsule.

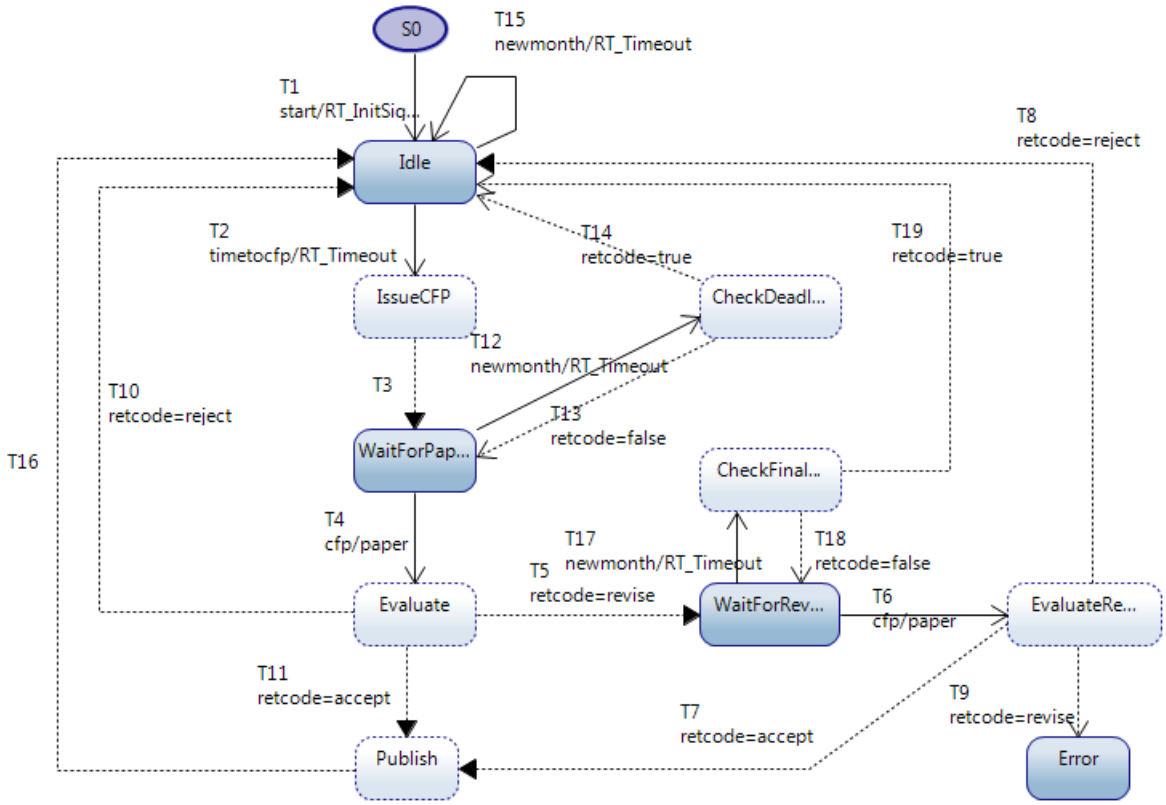

Figure 7: The Conference capsule.

signal is received from the ConferenceTimer, it moves to the IssueCFP from where it sends a call signal on the cfp port to the Academic1 capsule., and settles in the WaitForPapers state where it waits for papers on the cfp port. In this state, it also listens to timeout signals from the MonthTimer in order to check if the paper submission deadline has passed (in the CheckDeadline state). If the deadline has passed, it goes back to the Idle state. Otherwise, it remains in the WaitForPapers state. When a paper arrives on the cfp port, it is evaluated in the Evaluate state, which has an internal activity to perform the evaluation. The outcome might be: 1) to reject the paper, in which case it sends the reject signal to the academic via the cfp port; 2) to accept the paper, in which case it sends the accept signal to the academic via the cfp port and then it sends the paper to the Repository1 capsule via the papers port; or 3) to revise the paper, in which case it it sends the revise signal to the academic via the cfp port and then it waits for the revised version of the paper in the WaitForRevision state, again checking that it is within the deadline. In this later case, if the deadline passed, it goes back to the Idle state. If it receives the revision, it evaluates it and it can either reject it or accept it, with the same results as before.

The Vending Machine capsule The VendingMachine1 capsule has only one port: transaction with protocol Transaction (see Table 1 on page 3). It is used to interact with the Academic1 capsule.

The VendingMachine1 capsule's behaviour is shown in Figure 8 on page 8. Its behaviour is straightforward. In the WaitForMoney it waits for the insert_coin signal on its port, and when it is received, it evaluates whether more money is required or not in the IsEnough transient state. If more money is required, it sends back the signal more to the academic. If enough coins were inserted, it will send a change signal to the academic with the appropriate amount, and then it will go to the WaitForSelection state where it waits for the signals coffee_button or tea_button. Depending on which button is pushed, it will go to the corresponding state and send the signal coffee or tea to the academic, and then it will go back to the resting state.

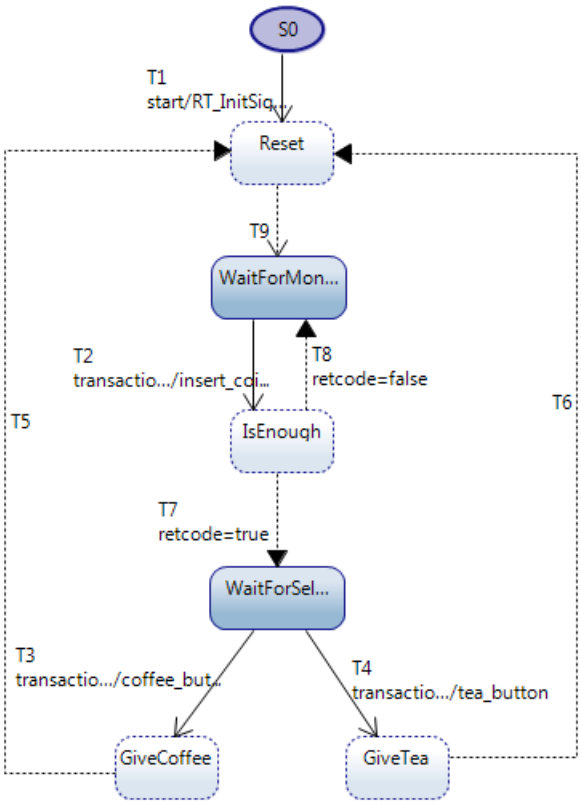

Figure 8: The Vending Machine capsule.

The Academic capsule The Academic1 capsule is the model's most complex. Its interface has three ports: grant with protocol ResearchGrant, cfp with protocol CallForPapers and drink with protocol Transaction (see Table 1 on page 3). The grant port is used to interact with the FundingAgency1 capsule, the cfp port is used to interact with the Conference1 capsule, and the drink port is used to interact with the VendingMachine1 capsule.

The Academic1 capsule's behaviour is shown in Figure 9 on page 10. The general idea is that the academic first tries to obtain funds, and when enough funds are available, it waits for a call for papers. Once it has been received, the academic uses some funds to obtain "fuel" (coffee or tea) from the vending machine, which determine the speed at which it will write papers. Then it submits the paper and may be asked to write a revision, for which it will try to procure more fuel.

In the FundsAvailable state, the academic checks whether he has funds (an internal attribute funds of the capsule). If it has enough funds, it goes directly to the WaitForCFP state to wait for the conference's call for papers. Otherwise, it applies for funds by sending a grant_application signal to the FundingAgency1 capsule through the grant port, and then in state WaitForGrant awaits a response on the same port. If it receives a reject signal, it goes to the Wait where it will wait one month (by listening to the timeout signal of the MonthTimer) and then it will try to reapply for a grant. If the grant was accepted, *i.e.*, if it received the signal funds on the grant port, it will come with an amount of money which will be added to the internal funds attribute, and then it will move on to the WaitForCFP state.

In the WaitForCFP state is will listen to the cfp port for call signals from the Conference1 capsule (while ignoring timeout signals from the MonthTimer). When a call signal arrives, it comes with a deadline and a page-limit, which are recorded as internal attributes for later use. Then the interaction with the VendingMachine1 capsule begins. The academic sends an insert_coin signal through the drink port and continues to do so while the signal more is received on the same port, and until the signal change arrives. The amount of change is added to the internal funds attribute. Then, the academic chooses a drink at random in the transient ChooseDrink state, and depending on the result of flipping a coin, it sends either the signal coffee_button or the signal tea_button, and waits, respectively, for the signal coffee or the signal tea to arrive. Depending on which of these arrives, an internal speed attribute, measured in number of pages written per month.

Once a drink has arrived, the academic gets to work on the Write state, where it will listen to the timeout signal from the MonthTimer, and check each month if the deadline has passed or not (in the CheckTime state). If the deadline passes before the paper is ready, it is abandoned, and the academic returns to the FundsAvailable state, where it will repeat the cycle. If the deadline has not passed but the academic has not reached the page limit, it will go back to the Write state and continue to work. If the deadline has not passed and the paper has reached the page limit, it submits the paper to the conference by sending a paper signal to the cfp port, and then waits for a response on the WaitForDecision state. If the decision signal arriving on the cfp port is revise, then it will go to the PayForDrink state and repeat the process of obtaining a drink and writing. If the decision signal was reject, it will go to the Sulk state, and then back to the FundsAvailable state to start all over. If the decision signal was accept, it will go to the DrinkChampagne state and send a publication signal back to the funding agency, to show that the money was used for research, and then it will go back to the FundsAvailable state to start all over.

Some requirements

We have presented a particular implementation of the capsules in this model, but the reader may be wondering what kind of requirements should such an application satisfy? The designer proceeds by adding components, which themselves may be refined further by introducing sub-components, and sub-sub-components, etc. This yields a hierarchical structure in which it is natural to associate each component with some requirements. Some typical requirements for various elements in this model could be the following:

(a) Whenever the academic applies for a grant to the funding agency, it will eventually get funds.

(b) Whenever the academic inserts enough coins in the vending machine, it will get coffee or tea.

(c) It is never the case that the academic submits a paper without having coffee or tea.

(d) It is never the case that the academic receives both coffee and tea.

(e) It is never the case that if the academic pushes the coffee button of the vending, it gets tea in return.

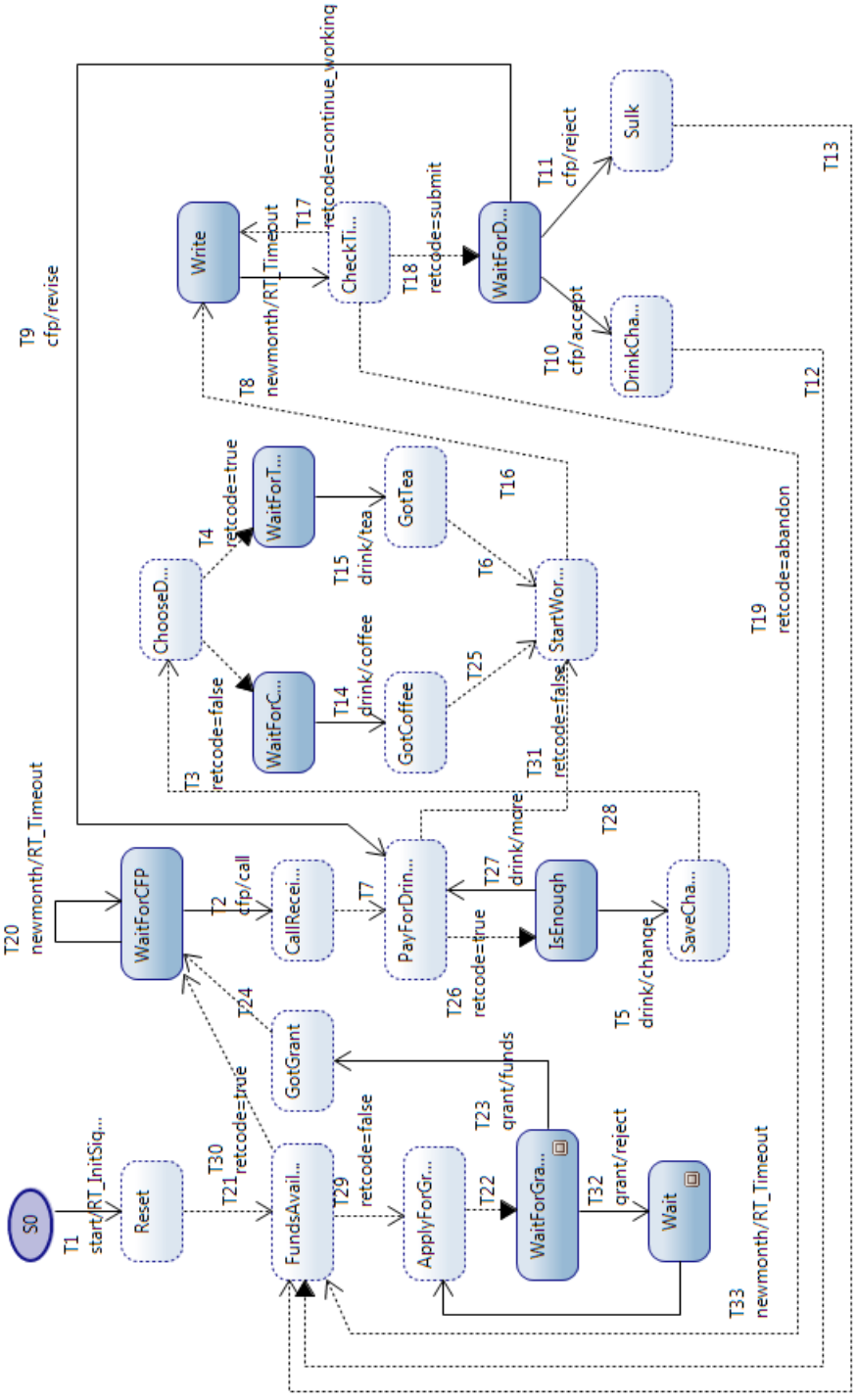

Figure 9: The Academic capsule.

(f) Whenever the academic receives a call for papers and has funds then it will obtain coffee or tea and submit a paper to the conference.

(g) Whenever the vending machine receives enough money it will issue coffee or tea.

(h) Whenever the conference issues a call for papers and receives a paper, then it will notify its author with an acceptance, rejection or revision notice and it accepted, the paper will be published in the repository.

(i) Whenever academia asks for a grant, if the funding agency has enough funds, it will grant funds to academia.

(j) It is always the case that the academic will eventually get a grant.

(k) It is always the case that the academic will eventually get a paper accepted at the conference.

(l) Whenever the funding agency issues funds to academia, eventually industry will receive papers if it requests them.

There are some important points to note. First, some of these requirements are *safety requirements*, *i.e.*, a undesirable event or behaviour does not ocurr (*e.g.*, paying for coffee and getting tea). Others are *liveness requirements*, *i.e.*, some desirable event or behaviour happens (*e.g.*, applying for a grant always results in receiving funds). Yet other requirements are *fairness requirements*, *i.e.*, some desirable behaviour is guaranteed to happen infinitely often (*e.g.*, getting grants or papers accepted). Note also that the last requirement may depend on academia satisfying some of the previous requirements.

These are the kind of requirements that an RTEdge^{TM} user might care about[1], and are the ones which our framework is intended to address. But now we will look at the relevant *general* questions.

1.2 The questions addressed in this project

Now we consider the scenarios and problems that we are trying to address. As explained in the Introduction, we are interested in compositional analysis using contract-based reasoning.

The general problem we are trying to address is a verification problem:

Question 1. *How do we know that a model satisfies its requirements?*

There is an implicit question here:

Question 2. *What kinds of requirements are we interested in?*

There are many kinds of requirements relevant to the context of concurrent, real-time embedded systems, such as performance requirements, schedulability and timing requirements, deployment requirements, cost, fault-tolerance or reliability requirements, etc. But we are interested in basic *functional requirements.*

Here we talk about functional requirements understood as the *behaviour* expected of a system, and in particular its *reactive behaviour*, and the relation between inputs and outputs in the system. In RTEdge^{TM}, the behaviour of a system, like in most concurrent, reactive and RTE systems, consists of a sequence of *events*, *actions* and *interactions* between components. The *external* or *observable behaviour* of a system, is often understood as a sequence of events, (input or output) messages or interactions that the component may have with its *environment*. Components can contain sub-components, and therefore, the environment of a component will include all other components with which the component under consideration may interact. These may be components within the system, or elements outside of the system. When we are talking about the observable behaviour of a component, we are abstracting away its *internal behaviour*, which may include interactions between its internal sub-components. Since we are interested, in this project, in functional requirements understood as the observable input/output behaviour of components, we abstract from other considerations such as the timing of events.[2]

[1] Not to mention academics.

[2] One can argue that in the context of real-time systems, the timing of events is fundamental to the correct functioning of the systems and therefore, functional requirements should include timing constraints. However, in the context of RTEdge^{TM},

Answer 1. (To Question 2) We are interested in *functional* requirements, understood as the set of observable behaviours of the component or model under scrutiny.

Having narrowed the focus of the problem to verifying functional requirements of RTEdge$^{\text{TM}}$ models, understood as observable behaviours, we could narrow the problem even more and consider only certain types of functional requirements, such as safety properties, liveness properties or fairness properties. However, at this point we do not need to commit ourselves to one any such subset of properties. Instead, we postpone this issue until it becomes necessary. At one level, the framework which we are adapting from [BDH$^+$12a] may be able deal with different kinds or (safety or liveness) properties under certain conditions, but clarifying those conditions is left as future work.

Question 1 does not arise in a vacuum. The verification issue arises because the user (the model designer) is creating a model and wishes to verify that it satisfies its requirements. This raises a question of pragmatics:

Question 3. *What steps should the user (i.e., the model designer) and the tool go through to perform the verification task?*

The answer to this question involves a series of activities to be done by the user or the tool, and these activities require us to address the following questions.

Question 4. *How to specify requirements of* RTEdge^{TM} *models?*

For this purpose, Edgewater has chosen the IEEE standard Property Specification Language (PSL).

Answer 2. (To Question 4) Requirements are to be specified using PSL.

A brief description of the core elements of PSL that we will use can be found in Appendix B.

Having chosen PSL as the specification language, the next question is how to use PSL to describe properties, specifications and contracts for RTEdge$^{\text{TM}}$ models. In particular we are interested in answering the following questions:

Question 5. *Can we use PSL as is, or will we need to adapt it to* RTEdge^{TM} *and to the assume/guarantee paradigm?*

Question 6. *Which constructs or elements of the* RTEdge^{TM} *language ought to be given a PSL specification?*

To answer these questions we have to consider the roles that the language constructs play in a model. We can summarize these roles, as currently used in RTEdge$^{\text{TM}}$ as follows:

- Protocols play a role analogous to the static type of ports, by restricting the kinds of signals allowed.[3]
- Interfaces play a role analogous to the static type of capsules, by restricting the interaction points of a capsule to a specific set of ports, and therefore, a specific set of signals as well. Interfaces can be seen as the "border" of capsules.
- Atomic capsules play the role of active agents defining concrete behaviour, as specified by a state machine.
- Composite capsules play the role of grouping capsules, structuring and hiding internal components and connections. By grouping capsules together and giving them a name and an interface, the composite can be used and re-used in other composite capsules, as if it was an atomic capsule.

We are interested in extending these roles as follows:

Edgewater has chosen to deal with the timing aspects as an issue orthogonal to functional requirements as described here, and timing is addressed by a separate schedulability analysis performed by the tool. As a consequence, the result of verification may be an over-approximation, reporting certain behaviours as valid or acceptable even when may not satisfy timing constraints and have been rejected by schedulability analysis.

[3] A note about terminology: in traditional signal theory, a signal is a function over a time domain, thus representing the evolution of some variable over time. In RTEdge$^{\text{TM}}$, however, a signal is rather a single datum, a token that can be communicated between capsules.

- Protocols should capture dynamic behaviour: a protocol should describe not only the set of signals allowed, but also the "conversations" or sequences of interactions and message exchanges allowed. For example, it should be possible to state that all legal conversations between the academic and the vending machine in Subsection 1.1 should have this form:

  > "Insert a coin, and possibly ask for more and receive more coins; then return some change, ask for the coffee button to be pushed, followed by issuing coffee, or ask for the tea button to be pushed, followed by issuing tea."
  >
  > In other words, the signal insert_coin is to be followed by zero or more repetitions of the more and insert_coin signals, followed by the change signal, followed by either coffee_button and then coffee or by tea_button and then tea.

- Interfaces should also describe permissible conversations of any capsule implementing the interface, but unlike protocols, these may refer to more than one port. In particular, it should be possible to describe what kinds of conversations are to be *guaranteed* by any capsule implementing the interface. For example, it should be possible to describe that the any capsule implementing the academic interface should have a behaviour guaranteeing:

  > "Whenever the academic accepts a call for papers, followed by the academic inserting coins on the vending machine, and pushing the coffee button and accepting coffee, or pushing the tea button and accepting tea, and eventually send out a paper to the conference."
  >
  > In other words, whenever signal call on port cfp is accepted, followed by the sending of insert_coin to port drink, as well as sending coffee_button and accepting coffee on port drink (resp. for tea_button, tea), then eventually signal paper will be sent to port cfp.

  Furthermore, it should be possible to describe *assumptions* on the environment of any capsule implementing the interface. For example, it should be possible to describe the following assumption for the academic:

  > "Whenever a grant application is sent out to the funding agency, the academic will eventually receive funds from the agency."
  >
  > In other words, whenever signal grant_application is sent to the grant port, eventually funds will be received at port grant.

- Atomic capsules should also describe legal conversations that they may have with their environment, and such conversations must comply with the capsule's interface. This is, the capsule may be annotated with a contract that should be a *refinement* of the contract associated to its interface.

- Composite capsules, just like atomic capsules, should also describe legal conversations that they may have with their environment, and such conversations must comply with the capsule's interface. This is, the capsule may be annotated with a contract that should be a *refinement* of the contract associated to its interface. Moreover, since a composite capsule may inherit from another composite capsule which has its own contract, the capsule's contract must also refine its parent capsule's contract. In any case, the contract for a composite capsule should only talk about its observable behaviour, and since it contains sub-components which may have connections hidden from the environment, the contract for the composite capsule can only talk about events on its interface and it cannot talk about internal events between sub-capsules. This is in accordance to the encapsulation principle.

Remark 1. There is a very subtle but fundamental point to be made about the specifications described above. Note that when discussing *guarantees* we talk about *accepting* (input) signals and when discussing *assumptions* we talk about *receiving* (input) signals. This is because a capsule can never guarantee that it will receive an input, as it does not excert control over the environment; the capsule can only guarantee that it will be in a state where it will be able to accept and consume the input. On the other hand, the reception of an input can be considered part of an assumption. In such case, we are assuming that the capsule's environment is sending it the corresponding signal. Similarly, the reference to outputs in a guarantee is subtly different than in an assumption. When we state that we are sending an output in a guarantee, it

can be interpreted as saying that the capsule indeed guarantees that it *will* send the output (not just that it *can* send the output), and this is so because in RTEdge™ sending is non-blocking, and is controlled by the capsule (it can *always* send any output, regardless of whether the environment is ready to accept it). However, when we mention an output in an assumption, we are making the assumption that the capsule's environment is willing to accept the message. It is essential for the designer to understand these differences when using the assume/guarantee paradigm as presented in this report.

At this point we can give a partial answer to Question 5 and Question 6. PSL, like any other propositional logic, is defined over a given set **AP** of *atomic propositions*. Typically, atomic propositions in PSL represent signals events or conditions over a state. In our case, when considering protocol specifications, atomic propositions can be taken to be the signals of the protocol, representing the presence of the signal on a given cycle or state. When considering contracts for interfaces and capsules we need to mention not only the signals in question but also the port in which they will ocurr (either as input or output) as well as their direction (input or output). Optionally we may also specify particular values carried by the signal.

Answer 3. (To Question 5 and Question 6) We will allow the user to annotate protocols with specifications of conversations allowed written as PSL specifications over the signals in the protocol. We will also allow the user to annotate interfaces, atomic and composite capsules with *contracts*, consisting of an *assumption* and a *guarantee*, written as PSL specifications over atomic propositions of the form

$$\langle direction \rangle : \langle port \rangle . \langle signal \rangle$$

A more detailed answer to Questions 4, 5 and 6 is given in Section 4.

Assuming that we have a way of specifying such requirements and contracts, the following question arise, regarding the relation between an interface and the protocols of its ports:

Question 7. *How do we know that the contract associated to an interface is compatible or consistent with the specifications associated to the protocols of its ports? In other words, how do we establish that an interface* complies *or* conforms *to its protocols?*

Similarly, we are faced with an analogous question regarding the relation between a capsule (atomic or composite) and its interface:

Question 8. *How do we know that a capsule's contract complies or conforms to the contract of its interface?*

And a similar question arises regarding interfaces or capsules which are related to each other via inheritance:

Question 9. *How do we know that the contract of an interface (resp. capsule) complies or conforms to the contract of its parent interface (resp. capsule)?*

Answering these questions requires us to define what we mean by compliance or conformance.

Once these questions have been answered, we come into the actual issue at hand, and consider Question 1 for some given RTEdge™ model. Assuming that we have answers for Questions 7, 8 and 9, we will have a mechanism to verify the consistency between the specifications and contracts between the protocols, interfaces and capsules in the model. Once such verification has been performed, we can turn our attention to verifying the behaviour of capsules themselves. To this end we make Question 1 more concrete with the following questions:

Question 10. *How do we establish that an atomic capsule satisfies its contract?*

And,

Question 11. *How do we establish that a composite capsule satisfies its contract?*

The **basic scenario** is as follows: we have a composite capsule K annotated with a contract C, and it has sub-components K_1, ..., K_n respectively annotated with contracts C_1, ..., C_n. So how do we know if K satisfies C? To answer this we first try to establish whether each sub-component K_i satisfies its contract C_i. This can be done by recursively applying the answers to Question 10 and Question 11. So if we have applied the analysis recursively and we have established that each part satisfies its contract, we can rephrase the question as follows:

Question 12. (Basic compositional inference problem) *How do we establish that a composite capsule K satisfies its contract C if we already know that each of its sub-components K_1, ..., K_n satisfy their respective contracts C_1, ..., C_n?*

One of the main advantages of answering this question is that if we have established tha K satisfies its contract C given that all its sub-components K_i satisfy theirs, then we can replace any sub-component K_j with any other component K'_j which also satisfies contract C_j and this will not affect the fact that the composition K satisfies C, *i.e.*, it is not necessary to re-run the analysis on the entire system; we only need to verify K'_j with respect to C_j. Furthermore, even if K'_j does not satisfy C_j, it may satisfy a contract C'_j. In this case, we may have to run the analysis required to answer Question 12 again, but we do not need to re-analyze the other sub-components K_i for $i \neq j$. The advantages of compositional analysis become clear. *Incremental analysis* becomes a reality as we are not required to re-analyze an entire system with every design change. We need to look only at the affected components.

Question 13. (Incremental analysis) *If a composite component K has been modified by adding, removing or changing a component, or by modifying a contract of a sub-component, or its own contract, how can we establish whether K still satisfies its contract without having to re-analyze unaffected components?*

Now, suppose that we have an answer to the basic compositional inference problem, and have used it to analyze a composite capsule K. Suppose that we have established that each of the sub-components K_i satisfies its contract C_i. But consider the case where the analysis resulted in showing that K *does not* satisfy its contract, even though its sub-components satisfied theirs. What can we do? We are confronted with a new scenario. One possible reason for a negative answer here might be that K was "missing" something. It may be that adding an additional component with certain behaviour will be enough to satisfy the contract. How do we find out what do we need to add to K so that it can now satisfy its contract? The designer may realize that K is missing a part X, so she adds a "*placeholder*" capsule X to be implemented, and the appropriate connections to the other sub-capsules in K. The problem is to find out an appropriate capsule implementation X to make K satisfy its contract. We refer to this problem as the **"missing part" scenario**, or the **"quotient problem"**.[4]

Question 14. (Quotient problem) *Given a composite capsule K with contract C and sub-capsules $K_1, ..., K_n$ with contracts $C_1, ..., C_n$ and a sub-capsule placeholder X:*

1. *What contract C_X should X have so that if we put, in place of X, a component implementation K_X that satisfies C_X and each K_i satisfies C_i then we can conclude that K satisfies C?*
2. *What should be an implementation of X that satisfies such contract C_X?*

Note that here we are looking for the *weakest*, i.e., the most general contract C_X that X should satisfy, in the sense that any other contract that can "complete" the specification would imply or would be a refinement of C_X. Finding the weakest such contract would give the designer more freedom to choose an implementation.

The answers[5]

The answer of Questions 3 through 14 will constitute the answer to Question 1. These answers, or partial answers are provided in the following sections as follows:

- An answer to Question 3 is suggested in Section 2.
- A more detailed answer to Questions 4, 5 and 6 is given in Section 4.
- A detailed answer to Questions 7, 8 and 9 is given in Section 5.
- A (partial) answer to Question 10 will be provided in Subsection 6.2.
- An answer to Question 12 (and therefore to Question 11) is given in Subsection 6.3.

[4] This is called the quotient problem by analogy with elementary algebra: given an equation of the form $a \cdot x = b$, its solution $x = b/a$ is the quotient between b and a.

[5] 42

- An answer to Question 13 is given in Section 7.
- An answer to Question 14 is given in Section 8.

2 User/tool work-flow

As stated in the introduction, the core of our proposed solution rests on the theoretical framework from [BDH+12a], which provides us answers to Question 12, the basic compositional inference problem, and suggests the basis to answer the quotient problem posed in Question 14, and a partial answer to Question 10. Questions 7, 8 and 9 are orthogonal. This is, the RTEdge™ platform can verify the compliance between protocols, interfaces and capsules independently of the compositional analysis itself. Hence, the answers to these questions we will give us a means to perform compliance verification and compositional analysis. Once we have such mechanisms implemented in the platform, we can define a possible workflow for the user and the tool.

Answer 4. (To Question 3) The general work-flow suggested for a designer and the tool would be as follows:

1. The user defines protocols, interfaces and capsules, annotating them with specifications (for protocols) and contracts (for interfaces and capsules). (Manual step)
2. Inter-element compliance verification: (Automatic step)
   (a) Whenever an interface is annotated with a contract, the tool would verify its compliance to the specifications of each of the ports's protocols, as well as the interface's parent's contract, if it has a parent.
   (b) Whenever a capsule is annotated with a contract, the tool would verify its conformance with the contract of the capsule's interface and the capsule's parent's contract, it if has a parent.
3. Compositional analysis: (Automatic step)
   (a) Sub-capsules are analyzed recursively. The base case of atomic capsules is done by a model-checker.
   (b) Compositional inference algorithm is applied.
4. If compositional analysis in step 3 fails, the designer has different options:
   (a) Modifying, adding or removing sub-components or contracts. (Manual step) Then,
      i. Inter-element compliance verification (as in step 2) is applied for all new elements added. (Automatic step).
      ii. Incremental analysis: (Automatic step).
         A. Analyze affected subcapsule recursively.
         B. Apply compositional inference algorithm.
   (b) Adding *capsule placeholders* and connections to the proper composite capsules. (Manual step) Then,
      i. Inter-element compliance verification (as in step 2) is applied for all new elements added. (Automatic step).
      ii. Quotient analysis: (Semi-automatic step)
         A. The quotient algorithm will produce the contract required by the new parts. (Automatic step)
         B. The user designs an implementation of the new parts (Manual step) or optionally, a tool may be used to automatically generate the skeleton of an implementation of the contract (Semi-automatic).
         C. For all manually designed new parts compositional analysis of step 3 may be required again.

The inter-element conformance verification of step 2 could be done on-the-fly, once the user has annotated the relevant elements, or it could be done by the user explicitly asking the tool to perform the required checks, but it might be preferable to automatically perform them before the compositional analysis of step 3 is performed. In other words, step 2 should be a prerequisite to step 3.

3 Theoretical framework for contract-based reasoning

In this section we transcribe a condensed version of the relevant elements from the contract-based theory from [BDH$^+$12a, BDH$^+$12b] and we extend it to support certain concepts that we'll need later. In this section, definitions, propositions and theorems from [BDH$^+$12a, BDH$^+$12b] are explicitly marked, and their proofs are available in [BDH$^+$12b]. Definitions, propositions and theorems not explicitly marked are our own, and their proofs are found in Subsection C.1. Subsequent sections show that we can apply this theoretical framework to our context. In Subsection C.2, C.3 and C.4 we prove the lemmas, propositions and theorems which are specific to our framework.

The core concept is that of a specification theory, which is essentially a family of specifications equipped with some operators, in particular with an operator to compose specifications, and a refinement relation that defines when a specification is more concrete or precise than another. In this generic framework, specifications can be different things such as automata, formulae in a temporal logic, regular expressions, sets of traces, or anything that satisfies the proper requirements. In Section 4, we will show that PSL constitutes a specification theory, but in this section we deal with specifications in general.

Definition 1 (**Specification theories [BDH$^+$12a]**)**.** A *(complete) specification theory* is a tuple $(\mathcal{S}, \otimes, /, \wedge, \leqslant)$ where

- $\mathcal{S}$ is a family of *specifications*
- $\otimes : \mathcal{S} \times \mathcal{S} \to \mathcal{S}$ is a *composition* operator[6]
- $/ : \mathcal{S} \times \mathcal{S} \to \mathcal{S}$ is a partial *quotient* operator
- $\wedge : \mathcal{S} \times \mathcal{S} \to \mathcal{S}$ is a partial *conjunction* operator
- $\leqslant \subseteq \mathcal{S} \times \mathcal{S}$ is a *refinement* preorder (*i.e.*, reflexive and transitive) preserved by composition:

  (A1) Whenever $P' \leqslant P$ and $Q' \leqslant Q$ then $P' \otimes Q' \leqslant P \otimes Q$

and where

- $/ : \mathcal{S} \times \mathcal{S} \to \mathcal{S}$ *must satisfy:*

  (A2) Q/P is defined if and only if $\exists X \in \mathcal{S}.\, P \otimes X \leqslant Q$

  (A3) If Q/P is defined, then $P \otimes (Q/P) \leqslant Q$

  (A4) If Q/P is defined, then $\forall X \in \mathcal{S}.\, P \otimes X \leqslant Q \;\to\; X \leqslant Q/P$

- $\wedge : \mathcal{S} \times \mathcal{S} \to \mathcal{S}$ must satisfy:

  (A5) $P \wedge Q$ is defined if and only if $\exists X.\, X \leqslant P \,\&\, X \leqslant Q$

  (A6) If $P \wedge Q$ is defined, then $P \wedge Q \leqslant P$ and $P \wedge Q \leqslant Q$

  (A7) If $P \wedge Q$ is defined, then $\forall X \in \mathcal{S}.\, X \leqslant P \,\&\, X \leqslant Q \;\to\; X \leqslant P \wedge Q$

Since specifications are meant to represent the behaviours of a system, the composition operator $\otimes$ provides a mechanism to make systems by combining (the specifications of) components: if P and Q are specifications, $P \otimes Q$ is a specification that combines P and Q. In particular contexts this will have certain particular semantics. For example, if the specifications are sets of traces, composition can be their union. If the specifications are automata, their composition may be their synchronous product.

The refinement relation between two specifications P and Q, denoted $P \leqslant Q$ normally represents the statement that P specifies fewer behaviours than Q or that P specifies at most the behaviours of Q. Hence, if $P \leqslant Q$, P is considered to be a more restricted specification than Q. A key property that this relation must satisfy is that is has to be preserved by composition (Axiom **(A1)**). This is critical, as it allows us to replace any specification P by any specification P' that refines P, while preserving the meaning of the context, (the

[6] *In* **[BDH$^+$12a]**, it is noted that composition can be partial, but we will consider only a total composition operator.

composite specification) in which P appears: suppose we have a complex specification $P \otimes Q$ and we have a refinement $P' \leqslant P$. Then, by Axiom **(A1)**, we have that $P' \otimes Q \leqslant P \otimes Q$, in other words, replacing P by P' results in a composite $P' \otimes Q$ which has less behaviours than $P \otimes Q$.

The quotient operator allows us to "complete a specification" so as to satisfy some requirements: suppose that we have some specifications P and Q but P alone does not refine Q. We want to be able to combine P with some other specification X so that their composition satisfies Q. This is, we need an X such that $P \otimes X \leqslant Q$. The quotient Q/P gives us the most general (the largest) specification that we can use in place of X to satisfy $P \otimes X \leqslant Q$.

The conjunction operator gives us another way of combining specifications. $P \wedge Q$ is the most general specification that satisfies both P and Q.

Definition 2 (**Specification equivalence [BDH+12a]**)**.** Given a complete specification theory $(\mathcal{S}, \otimes, /, \wedge, \leqslant)$, we say that two specifications $P, Q \in \mathcal{S}$ are *equivalent*, written $P = Q$ if $P \leqslant Q$ and $Q \leqslant P$.

By Definition 2, $=$ is symmetric, and since $\leqslant$ is a preorder relation, it is reflexive and transitive, so is $=$ which means that $=$ is an equivalence relation.

Proposition 1. *For any given complete specification theory $(\mathcal{S}, \otimes, /, \wedge, \leqslant)$, for any $X, P, P', Q, Q', R \in \mathcal{S}$:*

(i) if $P \wedge Q$ is defined and $X \leqslant P \wedge Q$ then $X \leqslant P$ and $X \leqslant Q$

(ii) if $P \wedge Q$ and $P' \wedge Q'$ are defined and $P' \leqslant P$ and $Q' \leqslant Q$ then $P' \wedge Q' \leqslant P \wedge Q$

(iii) if $P \wedge Q$ and $Q \wedge P$ are defined then $P \wedge Q = Q \wedge P$

(iv) if $P \wedge Q$, $Q \wedge R$, $P \wedge (Q \wedge R)$ and $(P \wedge Q) \wedge R$ are defined then $P \wedge (Q \wedge R) = (P \wedge Q) \wedge R$

(v) $P \wedge P = P$

(vi) if Q/P is defined and $X \leqslant Q/P$ then $P \otimes X \leqslant Q$

(vii) if $X \leqslant Y$ and $P \otimes Y \leqslant Q$ then $P \otimes X \leqslant Q$

In the remainder of this document whenever we make a statement using the conjuntion or the quotient partial operators, we will be making the implicit assumption that in the context of that statement, the corresponding conjunction or quotient specifications are defined. In other words, if we make a statement that uses $P \wedge Q$ or P/Q, the statement has the implicit condition "if $P \wedge Q$ (resp. P/Q) is defined ...".

Definition 3 (**Contracts; Definition 1 of [BDH+12a]**)**.** Given a complete specification theory $(\mathcal{S}, \otimes, /, \wedge, \leqslant)$, a *contract* is a pair $C = (A, G) \in \mathcal{S} \times \mathcal{S}$ where A is called the *assumption* and G is called the *guarantee*. We write

$$\mathsf{assumption}(C) = A$$

and

$$\mathsf{guarantee}(C) = G$$

Definition 4 (**Relativized refinement; Definition 2 of [BDH+12a]**)**.** Given a complete specification theory $(\mathcal{S}, \otimes, /, \wedge, \leqslant)$ and specifications $P, Q, R \in \mathcal{S}$ we say that P *refines* Q *relative to* (or *under the context*) R, written $P \leqslant_R Q$ if and only if for all $R' \in \mathcal{S}$, if $R' \leqslant R$ then $P \otimes R' \leqslant Q \otimes R'$.

Definition 5 (**Contract implementations and environments [BDH+12a]**)**.** Let $C = (A, G)$ be a contract. Then the set of *valid implementation specifications of* C is defined as

$$\mathsf{impl}[\![C]\!] \stackrel{def}{=} \{I \in \mathcal{S} \mid I \leqslant_A G\}$$

The set of *acceptable environment specifications* of C is defined as

$$\mathsf{env}[\![C]\!] \stackrel{def}{=} \{E \in \mathcal{S} \mid E \leqslant A\}$$

Definition 6 (**Contract refinement; Definition 4 of [BDH+12a]**)**.** Let $C = (A, G)$ and $C' = (A', G')$ be two contracts. We say that C' *refines* C, written $C' \preceq C$ if $\mathsf{impl}[\![C']\!] \subseteq \mathsf{impl}[\![C]\!]$ and $\mathsf{env}[\![C]\!] \subseteq \mathsf{env}[\![C']\!]$.

Proposition 2. *$\preceq$ is a preorder (i.e., a reflexive and transitive relation).*

Proposition 3. *Let C and C' be any contracts. For any specification I, if $I \in \mathsf{impl}[\![C]\!]$ and $C \preceq C'$ then $I \in \mathsf{impl}[\![C']\!]$.*

Theorem 1 (**Theorem 2 of [BDH$^+$12a]**). *Let $C = (A, G)$ and $C' = (A', G')$ be two contracts. Then $C' \preceq C$ if and only if $A \leqslant A'$ and $G' \leqslant_A G$.*

Definition 7 (**Contract equivalence**). Let $C = (A, G)$ and $C' = (A', G')$ be two contracts. We say that C and C' are *semantically equivalent*, written $C \simeq C'$ if $C \preceq C'$ and $C' \preceq C$. We say that C and C' are *strongly semantically equivalent*, written $C \equiv C'$ if $A = A'$ and $G = G'$ (see Definition 2).

Proposition 4. *Contract equivalence and strong contract equivalence are equivalence relations.*

Definition 8 (**Normal form; Definition 3 of [BDH$^+$12a]**). A contract $C = (A, G)$ is said to be in *normal form*, if for all implementations I, $I \leqslant_A G$ if and only if $I \leqslant G$, this is, if the implementation refines the guarantee independently of the assumptions.[7] We say that a contract $C = (A, G)$ *has normal form* $C' = (A', G')$ or that $C' = (A', G')$ is the normal form of $C = (A, G)$ if $C' = (A', G')$ is in normal form and and $C \simeq C'$.

Proposition 5. *For any contracts C, C',*

(i) if $C \equiv C'$ then $C \simeq C'$

(ii) if C and C' are in normal form then $C \simeq C'$ if and only if $C \equiv C'$

One of the goals of [BDH$^+$12a, BDH$^+$12b] is to provide a way to do compositional analysis: given a composite component K with subcomponents K_1 and K_2, we can try to establish if K satisfies a contract C by analyzing the sub-components separately to establish whether they satisfy their contracts C_1 and C_2, and then composing these contracts into a new contract $C_1 \boxtimes C_2$. If this contract $C_1 \boxtimes C_2$ refines C, then the original composite component K will satisfy C.

In order to define contract composition, we need the notion of contract dominance. Intuitively, a contract C dominates contracts C_1 and C_2 if (a) the composition of valid implementations of C_1 and C_2 is a valid implementation of C; and (b) the composition of any acceptable environment of C with any valid implementation of C_1 is an acceptable environment of C_2 (and viceversa with C_1 and C_2 swapped).

Definition 9 (**Contract domination; Definition 5 of [BDH$^+$12a]**). Given contracts C, C_1 and C_2, we say that C *dominates* C_1 and C_2 if:

(a) for all $I_1 \in \mathsf{impl}[\![C_1]\!]$ and all $I_2 \in \mathsf{impl}[\![C_2]\!]$, $I_1 \otimes I_2 \in \mathsf{impl}[\![C]\!]$

(b) for all $E \in \mathsf{env}[\![C]\!]$:

- for any $I_1 \in \mathsf{impl}[\![C_1]\!]$, $E \otimes I_1 \in \mathsf{env}[\![C_2]\!]$, and
- for any $I_2 \in \mathsf{impl}[\![C_2]\!]$, $E \otimes I_2 \in \mathsf{env}[\![C_1]\!]$

We say that contracts C_1 and C_2 are *dominatible* if there is a contract C that dominates them.

Definition 10 (**Contract composition; Definition 6 of [BDH$^+$12a]**). A contract C is called the *composition of contracts* C_1 and C_2 if

(a) C dominates C_1 and C_2, and

(b) for any contract C' that dominates C_1 and C_2, $C \preceq C'$.

In other words, the contract composition of two contracts is the least (most refined) contract that dominates them.

[7] The use of the term "normal form" as used in **[BDH$^+$12a]** may not agree with the way it is frequently used in other contexts such as logic and algebraic specifications. In those contexts the term "normal form" usually implies existence, uniqueness, and being an expression in some kind of formal language. None of these are necessarily the case in our context, but we will use the terminology of **[BDH$^+$12a]** for the sake of consistency.

Definition 11 (**Constructive contract composition; Definition 7 of [BDH$^+$12a]**)**.** Given two dominatible contracts $C_1 = (A_1, G_1)$ and $C_2 = (A_2, G_2)$ which have normal forms $\underline{C_1} = (\underline{A_1}, \underline{G_1})$ and $\underline{C_2} = (\underline{A_2}, \underline{G_2})$, we define

$$C_1 \boxtimes C_2 \stackrel{def}{=} (\widetilde{A}, \widetilde{G})$$

where

$$\widetilde{A} \stackrel{def}{=} (A_1/\underline{G_2}) \wedge (A_2/\underline{G_1})$$

and

$$\widetilde{G} \stackrel{def}{=} \underline{G_1} \otimes \underline{G_2}$$

The following theorem guarantees that the constructive composition is indeed a composition according to Definition 10.

Theorem 2 (**Theorem 4 of [BDH$^+$12a]**)**.** *If contracts C_1 and C_2 are dominatible then $C_1 \boxtimes C_2$ is (up to semantic equivalence) the composition of C_1 and C_2.*

Theorem 3 (**Theorem 6 of [BDH$^+$12a]**)**.** *Let C_1, C_2, D_1, D_2 be contracts with normal forms $\underline{C_1}$, $\underline{C_2}$, $\underline{D_1}$ and $\underline{D_2}$, and such that C_1 and C_2 are dominatible. If $D_1 \preceq C_1$ and $D_2 \preceq C_2$ then $\underline{D_1} \boxtimes \underline{D_2} \preceq \underline{C_1} \boxtimes \underline{C_2}$.*

Corollary 1. *Let C_1, C_2, D_1, D_2 be contracts with normal forms $\underline{C_1}$, $\underline{C_2}$, $\underline{D_1}$ and $\underline{D_2}$, and such that C_1 and C_2 are dominatible. If $D_1 \simeq C_1$ and $D_2 \simeq C_2$ then $\underline{D_1} \boxtimes \underline{D_2} \simeq \underline{C_1} \boxtimes \underline{C_2}$.*

We extend the notion of specification theory from [BDH$^+$12a] by considering theories where the composition operator is commutative and associative.

Definition 12 (**Commutative monoid specification theory**)**.** We say that the specification theory $(\mathcal{S}, \otimes, /, \wedge, \leqslant)$, is a *commutative monoid specification theory*, if $\otimes$ is commutative and associative with respect to $=$, *i.e.*, for all specifications $P, Q, R \in \mathcal{S}$,

(A8) $P \otimes Q = Q \otimes P$

(A9) $P \otimes (Q \otimes R) = (P \otimes Q) \otimes R$

We extend the notion of specification theory from [BDH$^+$12a] by considering theories where the composition shares some properties with conjunction.

Definition 13 (**Standard specification theory**)**.** A specification theory $(\mathcal{S}, \otimes, /, \wedge, \leqslant)$ is called *standard* if it satisfies the following addional axioms for composition:

(A10) $\forall X \in \mathcal{S}.\, X \leqslant P \,\&\, X \leqslant Q \rightarrow X \leqslant P \otimes Q$

(A11) $\forall X \in \mathcal{S}.\, P \leqslant Q \rightarrow P \otimes X \leqslant Q$

Axiom **(A10)** is analogous to Axiom **(A7)**. Intuitively it expresses that if a component X has less behaviours than P and Q, then it must have less behaviours than their composition $P \otimes Q$. Axiom **(A11)** is analogous to the weakening axiom in sequent calculus. Intuitively it expresses the idea that if a component P has less behaviours than a component Q, then composing P with some other component X will not add new behaviours.

Standard specification theories satisfy the following properties which will be useful later on.

Proposition 6. *Let $(\mathcal{S}, \otimes, /, \wedge, \leqslant)$ be a standard complete specification theory. Then, for all $X, P, Q \in \mathcal{S}$:*

(i) if $X \leqslant P \wedge Q$ then $X \leqslant P \otimes Q$

(ii) if $X \leqslant P$ and $X \leqslant Q/P$ then $X \leqslant Q$

(iii) $P \otimes Q \leqslant P$

(iv) if $P \leqslant Q$ then $P/R \leqslant Q/R$

(v) if $P \leqslant Q$ then $R/Q \leqslant R/P$

(vi) $(P/Q) \otimes R \leqslant (P \otimes R)/Q$

(vii) $P \wedge Q \leqslant Q/P$

Standard specification theories where the composition operator is commutative and associative yield some useful properties. These properties amount to simplifying the theory by identifying composition with conjunction resulting in a logical theory.

Definition 14 (**Simplified specification theory**)**.** A *simplified specification theory* is a standard complete specification theory which is also a commutative monoid specification theory.

Proposition 7. *Let $(\mathcal{S}, \otimes, /, \wedge, \leqslant)$ be a simplified specification theory. Then, for all $X, P, Q, R \in \mathcal{S}$:*

(i) $P \wedge Q = P \otimes Q$

(ii) $P \otimes Q \leqslant P$ *and* $P \otimes Q \leqslant Q$

(iii) $(R/Q)/P = R/(P \wedge Q)$

(iv) $(P \wedge Q)/R = (P/R) \wedge (Q/R)$

(v) $(Q/P)/P = Q/P$

(vi) $(P/Q) \otimes (Q/R) \leqslant P/R$

(vii) $(P/Q) \otimes (P/(Q \wedge R)) \leqslant P/(Q \wedge R)$

(viii) $(P/Q) \otimes (P'/Q') \leqslant (P \otimes P')/(Q \otimes Q')$

One of the advantages of these theories is that we obtain a simplified characterization of relativized refinement, as expressed in the following.

Lemma 1. *Let $(\mathcal{S}, \otimes, /, \wedge, \leqslant)$ be a simplified specification theory. Then, for all $P, Q, R \in \mathcal{S}$. Then $P \leqslant_R Q$ if and only if $P \otimes R \leqslant Q$.*

Another advantage of simplified specification theories is that every contract is guaranteed to have a semantically equivalent normal form.

Definition 15 (**Normalized contract**)**.** Let $(\mathcal{S}, \otimes, /, \wedge, \leqslant)$ be a simplified specification theory and let $C = (A, G)$ be any contract in $\mathcal{S}$. We call $\underline{C} \stackrel{def}{=} (A, \underline{G})$ the *normalized contract* of C, where $\underline{G} \stackrel{def}{=} G/A$.

Proposition 8. *Let $(\mathcal{S}, \otimes, /, \wedge, \leqslant)$ be a simplified specification theory and let $C = (A, G)$. Then*

(i) $C \simeq \underline{C}$

(ii) $\underline{C}$ *is in normal form.*

(iii) $\underline{\underline{C}} \equiv \underline{C}$

4 PSL Specifications and contracts

The basic problem of determining whether PSL is adequate in this context is now addressed. We begin by defining some notation that we will use throughout the report. The central claim (Theorem 4) is that PSL forms a complete specification theory in the sense of Section 3. The proof is given in Subsection C.2.

PSL is already a very rich specification language (see Appendix B). Nevertheless, we are interested in using PSL to describe properties of RTEdge^{TM} models, which means that we have to ensure that PSL expressions enjoy the appropriate vocabulary to talk about RTEdge^{TM} model elements. This entails endowing PSL with the means to refer to some of these elements. In general we are interested in describing properties about the functional behaviour of the model in terms of the interactions between components, that is, in terms of the signals exchanged between capsules over ports. Since signals are sent and received through ports, and they may carry data, the most basic statement that can be said is that a signal is an input or output on a specific port, carrying certain data. Other basic statements can describe constraints on a capsule's internal data attributes. These basic statements constitute the set of atomic propositions in our specification language, and all properties are built in terms of these atomic propositions using PSL's operators. In fact, we do not need to modify the syntax of PSL itself, as it is parametric on the set of atomic propositions to be used (see Appendix B). Hence it is enough for us to define the appropriate set of atomic propositions.

We now define formally such set of atomic propositions, but to do that we need certain sets to be defined.

Notation 1. We will assume the following sets:

- **Values**: the set of all possible data values that can be carried by signals.
- **Signals**: the set of all possible signal names.
- **Protocols**: the set of all possible protocol names.
- **Ports**: the set of all possible port names.
- **Connectors**: the set of all possible connector names.
- **Interfaces**: the set of all possible interface names.
- **AtomCapsules**: the set of all possible atomic capsule names.
- **CompCapsules**: the set of all possible composite capsule names.
- **Capsules**: the set of all possible capsule namess, *i.e.*, $\textbf{Capsules} \stackrel{def}{=} \textbf{AtomCapsules} \cup \textbf{CompCapsules}$.
- **Attributes**: the set of all possible attribute names.
- **Placeholders**: the set of all capsule *placeholder* names, *i.e.*, capsule-like entities with an interface but without an implementation.[8]
- **Components**: the set of all component names, *i.e.*, $\textbf{Components} \stackrel{def}{=} \textbf{Capsules} \cup \textbf{Placeholders}$.
- $\textbf{BoolExpr}_S$: the set of boolean expressions over the set S.

For any given set S we will write $S_\perp$ for the set $S \uplus \{\perp\}$ where $\perp$ represents a *null* or *empty value*, or *the absence* of a value.

Remark 2. The names of all elements, signals, protocols, ports, interfaces, capsules, attributes and connectors are assumed to be unique. In a real model this is not the case, but we can always replace all names with their fully qualified names: *e.g.*, in the example from Subsection 1.1, we have two connectors named conn1, one in the top-level application (Figure 1 on page 3), and one in capsule Academic1 (Figure 5 on page 6). These can be given unique names by replacing them with App.conn1 and App.Academic1.conn1 respectively.

[8] Place-holders are not a construct in RTEdge^{TM} per se. Rather they are "holes" in a model, which can be replaced by a capsule with the appropriate interface. Hence you can think of a capsule with place-holders as a template. Place-holders can also be thought of as meta-variables in the abstract syntax of capsules.

We now define the kinds of propositions that we may use in our specifications. We do this by defining the sets of atomic propositions for protocol specifications and the set of atomic propositions for component contracts.

Definition 16 (**Atomic propositions for specifications**)**.** We define the following sets:

- $\mathbf{ProtAP} \stackrel{def}{=} \mathbf{Signals} \times \mathbf{Values}_{\perp}$
- $\mathbf{AssumAP} \stackrel{def}{=} (\{\overline{\mathsf{in}}, \overline{\mathsf{out}}\} \times \mathbf{Ports} \times \mathbf{Signals} \times \mathbf{Values}_{\perp}) \uplus \mathbf{BoolExpr}_{\mathbf{Attributes}}$
- $\mathbf{GuaraAP} \stackrel{def}{=} (\{\mathsf{in}, \mathsf{out}\} \times \mathbf{Ports} \times \mathbf{Signals} \times \mathbf{Values}_{\perp}) \uplus \mathbf{BoolExpr}_{\mathbf{Attributes}}$
- $\mathbf{CompAP} \stackrel{def}{=} \mathbf{AssumAP} \cup \mathbf{GuaraAP}$

Notation 2. The null value $\perp$ will be used to denote optional items in the element, so we can leave it out.

- Any element $(s, v) \in \mathbf{ProtAP}$ can be written as $s(v)$ if $v \neq \perp$, or as s if $v = \perp$.
- Any element $(d, p, s, v) \in \mathbf{CompAP}$ will be written as

  $$d : p.s(v)$$

  if $v \neq \perp$, or as

  $$d : p.s$$

  if $v = \perp$. Hence:
  - $(\overline{\mathsf{in}}, p, s, v) \in \mathbf{AssumAP}$ will be written as

    $$\overline{\mathsf{in}} : p.s(v)$$

    if $v \neq \perp$, or as

    $$\overline{\mathsf{in}} : p.s$$

    if $v = \perp$.
  - $(\overline{\mathsf{out}}, p, s, v) \in \mathbf{AssumAP}$ will be written as

    $$\overline{\mathsf{out}} : p.s(v)$$

    if $v \neq \perp$, or as

    $$\overline{\mathsf{out}} : p.s$$

    if $v = \perp$.
  - $(\mathsf{in}, p, s, v) \in \mathbf{GuaraAP}$ will be written as

    $$\mathsf{in} : p.s(v)$$

    if $v \neq \perp$, or as

    $$\mathsf{in} : p.s$$

    if $v = \perp$.
  - $(\mathsf{out}, p, s, v) \in \mathbf{GuaraAP}$ will be written as

    $$\mathsf{out} : p.s(v)$$

    if $v \neq \perp$, or as

    $$\mathsf{out} : p.s$$

    if $v = \perp$.

Definition 17 (**PSL for** $\mathsf{RTEdge}^{\mathbf{TM}}$)**.** We define the following sets:

- **ProtPSL** is the set of PSL expressions over the set **ProtAP** of atomic propositions.
- **AssumPSL** is the set of PSL expressions over the set **AssumAP** of atomic propositions.
- **GuaraPSL** is the set of PSL expressions over the set **GuaraAP** of atomic propositions.
- **CompPSL** is the set of PSL expressions over the set **CompAP** of atomic propositions, *i.e.*, **CompPSL** = **AssumPSL** $\cup$ **GuaraPSL**.

The key or our approach is that PSL forms a specification theory in the sense of Section 3.

Theorem 4. *PSL is a simplified specification theory* $(\mathbf{CompPSL}, \otimes^{\mathsf{psl}}, /^{\mathsf{psl}}, \wedge^{\mathsf{psl}}, \leqslant^{\mathsf{psl}})$ *(cf. Definition 1) where:*

- $\mathcal{S} \stackrel{def}{=}$ **CompPSL** *is the set of PSL (Foundation Language) expressions*
- *Composition* $\otimes^{\mathsf{psl}}$ *is PSL conjunction:* $\varphi_1 \otimes^{\mathsf{psl}} \varphi_2 \stackrel{def}{=} \varphi_1 \wedge \varphi_2$
- *Quotient* $/^{\mathsf{psl}}$ *is PSL implication:* $\varphi_1 /^{\mathsf{psl}} \varphi_2 \stackrel{def}{=} \varphi_2 \rightarrow \varphi_1$
- *Conjunction* $\wedge^{\mathsf{psl}}$ *is PSL conjunction:* $\varphi_1 \wedge^{\mathsf{psl}} \varphi_2 \stackrel{def}{=} \varphi_1 \wedge \varphi_2$ *(where the right-hand side represents the PSL conjunction operator)*
- *Refinement is logical entailment:* $\varphi_1 \leqslant^{\mathsf{psl}} \varphi_2$ *iff* $\models \varphi_1 \rightarrow \varphi_2$

Definition 18 (**PSL Contracts**)**.** We define:

- A component *contract* is a pair (A, G) where $A, G \in$ **CompPSL**. We call **Contracts** $\stackrel{def}{=}$ **AssumPSL** $\times$ **GuaraPSL** the set of all component contracts.
- We say that *a contract* $C = (A, G)$ *is compatible with a component (interface, capsule or placeholder)* K if:
  - the port names ocurring in A or G is a subset of the ports of K (including inherited ports)
  - for each atomic proposition $(d, p, s, v) \in$ **CompAP**:
    - the signal s must be defined in the protocol of port p, and
    - the type of v must be compatible with the type of s (*e.g.*, the same type or a sub-type).
- An *annotated component* is a pair (K, C) where K is a component (interface, capsule or placeholder) and $C = (A, G)$ is a contract compatible with K.
- A *component-contract assignment* is a function

$$\mathsf{contract} : \mathbf{Components} \rightarrow \mathbf{Contracts}$$

that assigns a contract to each component in the model, so that $(K, \mathsf{contract}(K))$ is an annotated component. If

$$\mathsf{contract}(K) = (A, G)$$

we write

$$\mathsf{assumption}(K) = A$$

and

$$\mathsf{guarantee}(K) = G$$

Definition 19 (**Protocol specifications**)**.** We define:

- A *protocol specification* is an expression $S \in$ **ProtPSL**.
- We say that *a protocol specification* S *is compatible with a protocol* P if the signal names ocurring in S is a subset of the signals defined in P.

Spec	Direction	Point of view	Talks about	Meaning
Guarantee	$\mathsf{in} : p.s(v)$	capsule	capsule	capsule accepts/consumes input
	$\mathsf{out} : p.s(v)$	capsule	capsule	capsule sends output
Assumption	$\overline{\mathsf{in}} : p.s(v)$	capsule	environment	environment sends output, capsule receives input
	$\overline{\mathsf{out}} : p.s(v)$	capsule	environment	capsule sends output, environment accepts input
	$\mathsf{in} : p.s(v)$	environment	environment	capsule sends output, environment accepts input
	$\mathsf{out} : p.s(v)$	environment	environment	environment sends output, capsule receives input

Table 2: Meaning of in/out atomic propositions in assumptions and guarantees.

- An *annotated protocol* is a pair (P, S) where P is a protocol and $S \in \mathbf{ProtPSL}$ is a specification compatible with P.
- A *protocol-specification assignment* is a function

$$\mathsf{protspec} : \mathbf{Protocols} \to \mathbf{ProtPSL}$$

that assigns a specification to each protocol in the model, so that each pair $(P, \mathsf{protspec}(P))$ is an annotated protocol.

Answer 5. (To Question 5) Yes, PSL can be used as is, with the proviso that:

- PSL specifications for protocols use protocol signals as their atomic propositions.
- PSL assumptions and guarantees for interfaces and capsules use atomic propositions of the form:

$$\langle direction \rangle : \langle port \rangle . \langle signal \rangle [(\langle data \rangle)]$$

or boolean expressions over the capsule's attributes.

This will be enough for many purposes. However, it might be desirable to extend the syntax of PSL further to allow propositions of the form:

$$\langle direction \rangle : \langle port \rangle . \langle signal \rangle [(\langle variable(s) \rangle)]$$

This would enable the designer to express generic formulas to refer to the values carried by the signal in question, allowing the use of such variables elsewhere in the PSL expression. For example, the user could be allowed to express the following guarantee:

$$\mathsf{in} : \mathsf{grant.funds}(amount) \to \mathsf{F}\, \mathsf{out} : \mathsf{drink.insert\_coin}(amount/4)$$

Here we wish to express that whenever the academic receives *any* amount of funds through a grant, then eventually it will send a quarter of that amount to the vending machine. In this PSL-like expression, the second occurrence of the *amount* variable is *bound* to its introduction in the input proposition. Hence variables introduced by an input proposition would act as binders. This would also require of some syntax to distinguish such varaiables from access to a capsule's internal attributes. The addition of such binding variables, however, would entail an extension to the syntax and semantics of PSL. This may be considered in future work, and in the rest of this report we will assume only atomic propositions without variables.

Remark 3. When writing the $\langle direction \rangle$ part of an atomic proposition, the difference between in and $\overline{\mathsf{in}}$ and between out and $\overline{\mathsf{out}}$ respectively, depends on whether we are using them in an assumption or in a guarantee (cf. Remark 1):

- in and out can only be used in guarantee propositions.
- $\overline{\mathsf{in}}$ and $\overline{\mathsf{out}}$ can only be used in assumption propositions.
- The ocurrence of in in a guarantee entails that the component will *accept* and *consume* the signal.

- The occurrence of $\overline{\mathsf{in}}$ in an assumption entails that the component will receive the signal, *i.e.*, that the component's environment will send the signal to the component.
- The ocurrence of out in a guarantee entails that the component will send the output to the environment.
- The ocurrence of $\overline{\mathsf{out}}$ in an assumption entails that the environment is willing to accept the signal from the component.

The meaning of the input/output operators is summarized in Table 2 on page 26.

Answer 6. (To Question 6) Protocols, interfaces and capsules are all to be annotated with a specification as follows:

- Protocols: a protocol will be annotated with a PSL specification where atomic propositions are signals of the protocol. For example, the behaviour described above as:

  > "[...] the signal insert_coin is to be followed by zero or more repetitions of the more and insert_coin signals, followed by the change signal, followed by either coffee_button and then coffee or by tea_button and then tea."
  >
  > could be written as the following protocol specification:

  $$\mathsf{protspec}(Transaction) \stackrel{def}{=} \{\ \mathsf{insert\_coin}\,;\ \{\mathsf{more}\,;\mathsf{insert\_coin}\}\,[*]\,;\ \mathsf{change}\,;\ \{\{\mathsf{coffee\_button}\,;\mathsf{coffee}\}\,|\,\{\mathsf{tea\_button}\,;\mathsf{tea}\}\}\ \}\,[*]$$

  – Interfaces and capsules: they will be annotated with contracts of the form $C = (A, G)$ where A and G will be respectively, an assumption and a guarantee given as a PSL expressions over atomic propositions of the form described in Answer 5. For example, consider the following assumption for either the Academic interface or the Academic1 capsule:

  > "[...] whenever signal grant_application is sent to the grant port, eventually funds will be received at port grant."
  >
  > This could be written as:

  $$\mathsf{assumption}(Academic) \stackrel{def}{=} \mathsf{G}(\overline{\mathsf{out}} : \mathsf{grant.grant\_application} \rightarrow \mathsf{F}(\overline{\mathsf{in}} : \mathsf{grant.funds}))$$

  In such expressions, as described in Remark 3, the ocurrence of input atomic propositions such as $\overline{\mathsf{in}}$: grant.funds should be interpreted as the signal funds *will be received* on port grant, or in other words, an assumption that the capsule's environment will send that signal to the port. On the other hand, the ocurrence of output atomic expressions such as $\overline{\mathsf{out}}$: grant.grant_application should be interpreted as being true in a (global) state where the capsule's environment is willing to accept and consume the signal.
  Similarly, a guarantee such as:

  > "[...] whenever signal call on port cfp is accepted, followed by the sending of insert_coin to port drink, as well as sending coffee_button and accepting coffee on port drink (resp. for tea_button, tea), then eventually signal paper will be sent to port cfp."
  >
  > could be written as:

  $$\mathsf{guarantee}(Academic) \stackrel{def}{=} \mathsf{G}(\ \{\ \mathsf{in} : \mathsf{cfp.call}\,;\ \mathsf{out} : \mathsf{drink.insert\_coin}\,;\ \mathsf{out} : \mathsf{drink.coffee\_button}\,;\ \mathsf{in} : \mathsf{drink.coffee}\ \} \mapsto \mathsf{F}(\mathsf{out} : \mathsf{cfp.paper})\ )$$

  In this case, however, input and output atomic propositions are to be interpreted differently from the way they are interpreted in the assumptions. Here, an input atomic proposition such as

in : drink.coffee is true in a state where the academic will be willing to accept and consume such a signal, *i.e.*, it should be in a state with an outgoing transition that has the trigger for that port and signal. On the other hand, an output atomic proposition such as out : drink.insert_coin, should be interpreted as stating that the signal is sent.

– External tasks, proxy capsules and applications: these could be considered a special case. We see two alternatives:

  * Treat the application as a composite capsule that includes normal capsules and proxies. The interface of the application consists of all the OS ports in all of its proxies, and the external capsules are truly external. A contract for the application would then talk about conversations over OS ports only.
  * Treat the entire application as a single composite capsule, including normal capsules, as well as proxies and external tasks, where the last two are treated as normal. In this case, the application does no have any ports, as the external capsules would be inside.

  The second alternative does not seem very useful, as it would be unclear how to specify application-level requirements. Hence we favour the first alternative.

5 Conformance

A critical part of the user-workflow (Section 2) is to ensure the consistency or conformance between contracts of capsules, interfaces and protocol specifications. This is the essence of Questions 7, 8 and 9. In this Section we outline their answers.

5.1 Conformance between an interface and its protocols

A protocol specification defines the set of possible *conversations* or sequences of interactions allowed between two components. This sems to correpond to the *language* of an expression in the sense of automata theory with inputs and outputs as the alphabet in the language. In our case, however, we are using PSL specifications, and the meaning of these is similar to the traditional notion of language except for a subtle but significant difference: the alphabet of languages defined by PSL expressions consist of *sets of atomic propositions*.

Protocol specifications are given as PSL expressions, and the semantics of PSL is given in terms of sequences satisfied by an expression. This is, the formal semantics of PSL (see [IEE12a]) defines a satisfability relation $\models$ between sequences and PSL formulas: the notation

$$v \models \varphi$$

means that the PSL formula φ *holds* in the sequence $v = v_0 v_1 ...$ or that v *satisfies* φ, where the items v_i of a sequence $v = v_0 v_1 ...$ are sets of atomic propositions: $\forall i \geqslant 0.\, v_i \in 2^{\mathbf{AP}}$. Informally, each set v_i contains all atomic propositions which are true at that point in time, where time is understood as consisting of discrete steps, so $p \in v_i$ if p is true in the i-th step.

Then, the language of a PSL expression is defined as

$$\mathcal{L}_{\mathbf{PSL}}(\varphi) \stackrel{def}{=} \{v \,|\, v \models \varphi\}$$

To see how this is different from the notion of the language of a regular expression, consider the PSL SERE "$\{a\,;b\}$" and the regular expression "ab". We have that the language of "ab" is $\mathcal{L}_{\mathbf{regexp}}(ab) = \{ab\}$, which consist of the single string ab. On the other hand, $\mathcal{L}_{\mathbf{PSL}}(\{a\,;b\}) = \{v_0 v_1 ... \,|\, a \in v_0, b \in v_1\}$. Hence this language contains an infinite number of strings, each of which has the form $(\{a\} \cup A_0)(\{b\} \cup A_1)....$ So the regular expression represents a very strict requirement that the first item in the sequence *must be* a (and nothing else) and the second item in the sequence *must be* b and nothing else. Compare this to the PSL expression which makes a weaker statement: the first item in a sequence must contain a, or equivalently, a must hold in the first item (but other atomic propositions may hold as well), and the second item must contain b (b must be true in the second step).

Question 7 can be phrased as follows: given an interface F annotated with a contract $C = (A, G)$, and a protocol R annotated with a specification S, how do we establish that C *conforms* to S?

We are looking to define the criteria necessary for such conformance. Intuitively, we would expect that the conversations (behaviours) specified by A (resp. G) ought to be allowed by S, so the set of all possible sequences specified by A (resp. G) should be a subset of the sequences specified by S. This, hoever, is not the case in general. The reason is that they do not "speak the same language", this is, the set of all possible atomic propositions in A (or G) is not the same as the set of all possible atomic propositions in S. To see this, consider the following guarantee for the Academic interface:

$$\begin{aligned} \mathsf{guarantee}(\mathit{Academic}) \stackrel{def}{=} \mathsf{G}(\;& \{\, \mathsf{in} : \mathsf{cfp.call}\,; \\ & \quad \mathsf{out} : \mathsf{drink.insert\_coin}\,; \\ & \quad \mathsf{out} : \mathsf{drink.coffee\_button}\,\} \\ & \mapsto \mathsf{F}(\mathsf{in} : \mathsf{drink.coffee} \wedge \mathsf{X}\,\mathsf{out} : \mathsf{cfp.paper}) \quad) \end{aligned}$$

and the following protocol specification for the Transation protocol:

$$\begin{aligned} \mathsf{protspec}(\mathit{Transaction}) \stackrel{def}{=} \{ \;& \mathsf{insert\_coin}\,; \\ & \{\mathsf{more}\,; \mathsf{insert\_coin}\}\,[^*]\,; \\ & \mathsf{change}\,; \\ & \{\{\mathsf{coffee\_button}\,; \mathsf{coffee}\} \,|\, \{\mathsf{tea\_button}\,; \mathsf{tea}\}\} \quad \}\,[^*] \end{aligned}$$

Several problems arise. $\mathcal{L}_{\mathbf{PSL}}(Academic)$ will include all sequences $v = v_0v_1v_2...v_kv_{k+1}...$ where $\mathsf{in} : \mathsf{cfp.call} \in v_0$, $\mathsf{out} : \mathsf{drink.insert\_coin} \in v_1$, $\mathsf{out} : \mathsf{drink.coffee\_button} \in v_2$, $\mathsf{in} : \mathsf{drink.coffee} \in v_k$ and $\mathsf{out} : \mathsf{cfp.paper} \in v_{k+1}$ for some $k > 2$. On the othe hand, $\mathcal{L}_{\mathbf{PSL}}(Transaction)$ will include all sequences $u = u_0u_1...$ where $\mathsf{insert\_coin} \in u_0$, etc. The first problem becomes apparent. The set of atomic propositions of a guarantee are of the form $\mathsf{in} : p.s(v)$ or $\mathsf{out} : p.s(v)$ (and similarly for assumptions). On the other hand, the set of atomic propositions of a protocol specification are of the form s where s is a signal name. Hence the formulas cannot be compared or combined directly.

In order to solve the problem we need to make specifications and contracts "speak the same language", or rather, share the same alphabet, this is, we need to rewrite them in such a way that they have the same set of atomic propositions so that we can establish if one formula implies the other. To this end we define a few functions that will perform this rewriting.

First, we need to define a function pproj which takes as input a protocol specification $S \in \mathbf{ProtPSL}$, a component $K \in \mathbf{Components}$ and a port $p \in \mathbf{Ports}$ and returns the "projection" of the specification onto that component's port, this is, it translated the protocol from the point of view of a connector to the point of view of the capsule and the port. Essentially this entails replacing every signal $s(v)$ in the protocol specification with $\mathsf{in} : p.s(v)$ if p is a base port in K and s is an input signal of the protocol, or if p is a conjugate port and s is an output signal of the protocol; or replace it with $\mathsf{out} : p.s(v)$ if if p is a base port in K and s is an output signal of the protocol, or if p is a conjugate port and s is an input signal of the protocol.

Definition 20 (**Protocol specification port projection**)**.** We define the function $\mathsf{pproj} : \mathbf{ProtPSL} \to \mathbf{Protocols} \to \mathbf{Interfaces} \to \mathbf{Ports} \to \mathbf{CompPSL}$ be as shown in Figure 10 on page 31. We write $\mathsf{pproj}[\![S]\!]_{R|F.q}$ for $\mathsf{pproj}(S)(R)(F)(q)$, *i.e.*, the result of applying the function pproj to the protocol specification $S \in \mathbf{ProtPSL}$, of the protocol $R \in \mathbf{Protocols}$, onto port $q \in \mathbf{Ports}$ of interface $F \in \mathbf{Interfaces}$.

Going back to the example, we see that applying this projection on the specification of the Transaction protocol yields

$\mathsf{pproj}[\![\mathsf{protspec}(Transaction)]\!]_{\mathsf{Transaction|Academic.drink}} =$

```
{   in : drink.insert_coin ;
    {out : drink.more ; in : drink.insert_coin} [*] ;
    out : drink.change ;
    {{in : drink.coffee_button ; out : drink.coffee} | {in : drink.tea_button ; out : drink.tea}}   } [*]
```

Now we define a function flip which takes an assumption expression $A \in \mathbf{AssumPSL}$ and flips the direction of signals in the assumption. Informally this translates assumptions written from the point of view of a component into guarantees of the component's environment.

Definition 21 (**Assumption direction flipping**)**.** Let the function $\mathsf{flip} : \mathbf{AssumPSL} \to \mathbf{GuaraPSL}$ as showin in Figure 11 on page 32. We write $\mathsf{flip}[\![A]\!]$ for $\mathsf{flip}(A)$ where $A \in \mathbf{AssumPSL}$ is a PSL assumption.

With these renamings we can define the criteria for conformance between an interface's contract and a protocol's behaviour. This can be reduced to checking the *validity* of certain PSL formulas. A PSL formula φ is said to be *valid*, written $\vDash \varphi$ if $v \vDash \varphi$ for all sequences v.

Definition 22 (**Interface/protocol strict conformance**)**.** Given an interface F annotated with a contract $C = (A, G)$, and a protocol R annotated with a specification S we say that F *conforms strictly to* R if for all ports $p \in \mathsf{ports}(F)$:

1. $\mathcal{L}_{\mathbf{PSL}}(G) \subseteq \mathcal{L}_{\mathbf{PSL}}(\mathsf{pproj}[\![S]\!]_{R|F.p})$, and
2. $\mathcal{L}_{\mathbf{PSL}}(\mathsf{flip}[\![A]\!]) \subseteq \mathcal{L}_{\mathbf{PSL}}(\mathsf{pproj}[\![S]\!]_{R|F.p})$

The previous definition gives us an intuitive criterion for what do we mean by a contract conforming to the specification of a protocol. However, we need an actionable characterization which can be checked algorithmically. It turns out that language inclusion for PSL corresponds to logical entailment between the corresponding PSL formulas. This gives us a logical characterization for the definition of interface/protocol conformance which can be determined using PSL validity checking. The following Theorem states such characterization.

$\mathsf{pproj}[\![s(v)]\!]_{R\|F.p}$	$\stackrel{def}{=}$	$\mathsf{in} : p.s(v)$	if $\mathsf{kindmap}(F)(p) = \mathsf{base}$ and $s \in \mathsf{isignals}(R)$ or $\mathsf{kindmap}(F)(p) = \mathsf{conj}$ and $s \in \mathsf{osignals}(R)$
$\mathsf{pproj}[\![s(v)]\!]_{R\|F.p}$	$\stackrel{def}{=}$	$\mathsf{out} : p.s(v)$	if $\mathsf{kindmap}(F)(p) = \mathsf{base}$ and $s \in \mathsf{osignals}(R)$ or $\mathsf{kindmap}(F)(p) = \mathsf{conj}$ and $s \in \mathsf{isignals}(R)$
$\mathsf{pproj}[\![f(a)]\!]_{R\|F.p}$	$\stackrel{def}{=}$	$f(a)$	if $f(a) \in \mathbf{BoolExpr}_{\mathbf{Attributes}}$
$\mathsf{pproj}[\![!b]\!]_{R\|F.p}$	$\stackrel{def}{=}$	$!\mathsf{pproj}[\![b]\!]_{R\|F.p}$	if $b \in \mathbf{BoolExpr}$
$\mathsf{pproj}[\![b_1 \wedge b_2]\!]_{R\|F.p}$	$\stackrel{def}{=}$	$\mathsf{pproj}[\![b_1]\!]_{R\|F.p} \wedge \mathsf{pproj}[\![b_2]\!]_{R\|F.p}$	if $b_1, b_2 \in \mathbf{BoolExpr}$
$\mathsf{pproj}[\![\{r\}]\!]_{R\|F.p}$	$\stackrel{def}{=}$	$\{\mathsf{pproj}[\![r]\!]_{R\|F.p}\}$	if $r \in \mathbf{SERE}$
$\mathsf{pproj}[\![r_1 ; r_2]\!]_{R\|F.p}$	$\stackrel{def}{=}$	$\{\mathsf{pproj}[\![r_1]\!]_{R\|F.p}\} ; \{\mathsf{pproj}[\![r_2]\!]_{R\|F.p}\}$	if $r_1, r_2 \in \mathbf{SERE}$
$\mathsf{pproj}[\![r_1 : r_2]\!]_{R\|F.p}$	$\stackrel{def}{=}$	$\{\mathsf{pproj}[\![r_1]\!]_{R\|F.p}\} : \{\mathsf{pproj}[\![r_2]\!]_{R\|F.p}\}$	if $r_1, r_2 \in \mathbf{SERE}$
$\mathsf{pproj}[\![r_1 \mid r_2]\!]_{R\|F.p}$	$\stackrel{def}{=}$	$\{\mathsf{pproj}[\![r_1]\!]_{R\|F.p}\} \mid \{\mathsf{pproj}[\![r_2]\!]_{R\|F.p}\}$	if $r_1, r_2 \in \mathbf{SERE}$
$\mathsf{pproj}[\![r_1 \,\&\&\, r_2]\!]_{R\|F.p}$	$\stackrel{def}{=}$	$\{\mathsf{pproj}[\![r_1]\!]_{R\|F.p}\} \,\&\&\, \{\mathsf{pproj}[\![r_2]\!]_{R\|F.p}\}$	if $r_1, r_2 \in \mathbf{SERE}$
$\mathsf{pproj}[\![r\,[*0]]\!]_{R\|F.p}$	$\stackrel{def}{=}$	$\{\mathsf{pproj}[\![r]\!]_{R\|F.p}\}\,[*0]$	if $r \in \mathbf{SERE}$
$\mathsf{pproj}[\![r\,[*]]\!]_{R\|F.p}$	$\stackrel{def}{=}$	$\{\mathsf{pproj}[\![r]\!]_{R\|F.p}\}\,[*]$	if $r \in \mathbf{SERE}$
$\mathsf{pproj}[\![b!]\!]_{R\|F.p}$	$\stackrel{def}{=}$	$\mathsf{pproj}[\![b]\!]_{R\|F.p}!$	if $b \in \mathbf{BoolExpr}$
$\mathsf{pproj}[\![(\varphi)]\!]_{R\|F.p}$	$\stackrel{def}{=}$	$(\mathsf{pproj}[\![\varphi]\!]_{R\|F.p})$	if $\varphi \in \mathbf{FL}$
$\mathsf{pproj}[\![\neg\varphi]\!]_{R\|F.p}$	$\stackrel{def}{=}$	$\neg\mathsf{pproj}[\![\varphi]\!]_{R\|F.p}$	if $\varphi \in \mathbf{FL}$
$\mathsf{pproj}[\![\varphi_1 \wedge \varphi_2]\!]_{R\|F.p}$	$\stackrel{def}{=}$	$(\mathsf{pproj}[\![\varphi_1]\!]_{R\|F.p}) \wedge (\mathsf{pproj}[\![\varphi_2]\!]_{R\|F.p})$	if $\varphi_1, \varphi_2 \in \mathbf{FL}$
$\mathsf{pproj}[\![\mathsf{X!}\,\varphi]\!]_{R\|F.p}$	$\stackrel{def}{=}$	$\mathsf{X!}(\mathsf{pproj}[\![\varphi]\!]_{R\|F.p})$	if $\varphi \in \mathbf{FL}$
$\mathsf{pproj}[\![[\varphi_1 \,\mathsf{U}\, \varphi_2]]\!]_{R\|F.p}$	$\stackrel{def}{=}$	$[(\mathsf{pproj}[\![\varphi_1]\!]_{R\|F.p})\,\mathsf{U}(\mathsf{pproj}[\![\varphi_1]\!]_{R\|F.p})]$	if $\varphi_1, \varphi_2 \in \mathbf{FL}$
$\mathsf{pproj}[\![\varphi\,\mathsf{abort}\,b]\!]_{R\|F.p}$	$\stackrel{def}{=}$	$(\mathsf{pproj}[\![\varphi]\!]_{R\|F.p})\,\mathsf{abort}(\mathsf{pproj}[\![b]\!]_{R\|F.p})$	if $\varphi \in \mathbf{FL}, b \in \mathbf{BoolExpr}$
$\mathsf{pproj}[\![r \mapsto \varphi]\!]_{R\|F.p}$	$\stackrel{def}{=}$	$(\mathsf{pproj}[\![r]\!]_{R\|F.p}) \mapsto (\mathsf{pproj}[\![\varphi]\!]_{R\|F.p})$	if $\varphi \in \mathbf{FL}, r \in \mathbf{SERE}$
$\mathsf{pproj}[\![r!]\!]_{R\|F.p}$	$\stackrel{def}{=}$	$\{\mathsf{pproj}[\![r]\!]_{R\|F.p}\}!$	if $r \in \mathbf{SERE}$

Figure 10: Protocol specification port projection.

$$
\begin{array}{rcll}
\mathsf{flip}[\![\overline{\mathsf{in}} : p.s(v)]\!] & \stackrel{def}{=} & \mathsf{out} : p.s(v) & \\
\mathsf{flip}[\![\overline{\mathsf{out}} : p.s(v)]\!] & \stackrel{def}{=} & \mathsf{in} : p.s(v) & \\
\mathsf{flip}[\![f(a)]\!] & \stackrel{def}{=} & f(a) & \text{if } f(a) \in \mathbf{BoolExpr}_{\mathbf{Attributes}} \\
\mathsf{flip}[\![!b]\!] & \stackrel{def}{=} & !\mathsf{flip}[\![b]\!] & \text{if } b \in \mathbf{BoolExpr} \\
\mathsf{flip}[\![b_1 \wedge b_2]\!] & \stackrel{def}{=} & \mathsf{flip}[\![b_1]\!] \wedge \mathsf{flip}[\![b_2]\!] & \text{if } b_1, b_2 \in \mathbf{BoolExpr} \\
\mathsf{flip}[\![\{r\}]\!] & \stackrel{def}{=} & \{\mathsf{flip}[\![r]\!]\} & \text{if } r \in \mathbf{SERE} \\
\mathsf{flip}[\![r_1 \,;\, r_2]\!] & \stackrel{def}{=} & \{\mathsf{flip}[\![r_1]\!]\} \,;\, \{\mathsf{flip}[\![r_2]\!]\} & \text{if } r_1, r_2 \in \mathbf{SERE} \\
\mathsf{flip}[\![r_1 : r_2]\!] & \stackrel{def}{=} & \{\mathsf{flip}[\![r_1]\!]\} : \{\mathsf{flip}[\![r_2]\!]\} & \text{if } r_1, r_2 \in \mathbf{SERE} \\
\mathsf{flip}[\![r_1 \mid r_2]\!] & \stackrel{def}{=} & \{\mathsf{flip}[\![r_1]\!]\} \mid \{\mathsf{flip}[\![r_2]\!]\} & \text{if } r_1, r_2 \in \mathbf{SERE} \\
\mathsf{flip}[\![r_1 \,\&\&\, r_2]\!] & \stackrel{def}{=} & \{\mathsf{flip}[\![r_1]\!]\} \,\&\&\, \{\mathsf{flip}[\![r_2]\!]\} & \text{if } r_1, r_2 \in \mathbf{SERE} \\
\mathsf{flip}[\![r\,[*0]]\!] & \stackrel{def}{=} & \{\mathsf{flip}[\![r]\!]\}\,[*0] & \text{if } r \in \mathbf{SERE} \\
\mathsf{flip}[\![r\,[*]]\!] & \stackrel{def}{=} & \{\mathsf{flip}[\![r]\!]\}\,[*] & \text{if } r \in \mathbf{SERE} \\
\mathsf{flip}[\![b!]\!] & \stackrel{def}{=} & \mathsf{flip}[\![b]\!]! & \text{if } b \in \mathbf{BoolExpr} \\
\mathsf{flip}[\![(\varphi)]\!] & \stackrel{def}{=} & (\mathsf{flip}[\![\varphi]\!]) & \text{if } \varphi \in \mathbf{FL} \\
\mathsf{flip}[\![\neg\varphi]\!] & \stackrel{def}{=} & \neg\mathsf{flip}[\![\varphi]\!] & \text{if } \varphi \in \mathbf{FL} \\
\mathsf{flip}[\![\varphi_1 \wedge \varphi_2]\!] & \stackrel{def}{=} & (\mathsf{flip}[\![\varphi_1]\!]) \wedge (\mathsf{flip}[\![\varphi_2]\!]) & \text{if } \varphi_1, \varphi_2 \in \mathbf{FL} \\
\mathsf{flip}[\![\mathsf{X!}\,\varphi]\!] & \stackrel{def}{=} & \mathsf{X!}(\mathsf{flip}[\![\varphi]\!]) & \text{if } \varphi \in \mathbf{FL} \\
\mathsf{flip}[\![[\varphi_1 \,\mathsf{U}\, \varphi_2]]\!] & \stackrel{def}{=} & [(\mathsf{flip}[\![\varphi_1]\!])\,\mathsf{U}(\mathsf{flip}[\![\varphi_1]\!])] & \text{if } \varphi_1, \varphi_2 \in \mathbf{FL} \\
\mathsf{flip}[\![\varphi\,\mathsf{abort}\,b]\!] & \stackrel{def}{=} & (\mathsf{flip}[\![\varphi]\!])\,\mathsf{abort}(\mathsf{flip}[\![b]\!]) & \text{if } \varphi \in \mathbf{FL}, b \in \mathbf{BoolExpr} \\
\mathsf{flip}[\![r \mapsto \varphi]\!] & \stackrel{def}{=} & (\mathsf{flip}[\![r]\!]) \mapsto (\mathsf{flip}[\![\varphi]\!]) & \text{if } \varphi \in \mathbf{FL}, r \in \mathbf{SERE} \\
\mathsf{flip}[\![r!]\!] & \stackrel{def}{=} & \{\mathsf{flip}[\![r]\!]\}! & \text{if } r \in \mathbf{SERE}
\end{array}
$$

Figure 11: Assumption direction flipping.

Algorithm 1 Interface/protocol conformance.

Require: $F = (P, L, prot, kind)$ is an interface with a collection of ports P and protocols L, annotated with a contract $C = (A, G)$

Ensure: returns true if C conforms to all port's specifications

```
1:  function CHECK-INTERFACE-PROTOCOL-CONFORMANCE(F)
2:      C ← contract(F)
3:      A ← assumption(C)
4:      G ← guarantee(C)
5:      P ← ports(F)
6:      prot ← protmap(F)
7:      for all p ∈ P do
8:          R ← prot(p)                          ▷ Let R be the protocol of port p
9:          S ← protspec(R)                      ▷ Let S be R's protocol PSL specification
10:         Y ← pproj⟦S⟧_{R|F.p}                 ▷ Let Y be the pojection of the protocol spec into the port
11:         f1 ← (G → Y)
12:         r1 ← CHECK-VALID-PSL(f1)
13:         if r1 is false then
14:             return false
15:         else
16:             f2 ← (flip⟦A⟧ → Y)
17:             r2 ← CHECK-VALID-PSL(f2)
18:             return r2
19:         end if
20:     end for
21: end function
```

Theorem 5. *Given an interface F annotated with a contract $C = (A, G)$, and a protocol R annotated with a specification S, F conforms* strictly *to R if and only if for all ports $p \in \mathsf{ports}(F)$:*

1. $\models G \rightarrow \mathsf{pproj}[\![S]\!]_{R|F.p}$, *and*
2. $\models \mathsf{flip}[\![A]\!] \rightarrow \mathsf{pproj}[\![S]\!]_{R|F.p}$

Answer 7. (To Question 7) We establish the conformance of an interface's contract to a protocol's specificaction according to the criteria given in Theorem 5. This is, we can run Algorithm 1. Like many of the algorithms presented in this report, whenever we check $\models \varphi$ for some PSL formula φ, we are assuming an external tool that checks for validity of PSL formulas. Such tool is invoked by the CHECK-VALID-PSL function.

If we go back to the guarantee for the Academic interface and the Transaction protocol at the beginning of this section, we will see that Academic does not conform to Transaction. But this is because the Transaction protocol projection $\mathsf{pproj}[\![\mathsf{protspec}(Transaction)]\!]_{\mathsf{Transaction|Academic.drink}}$ expects a "in : drink.insert_coin" in the first cycle of any conversation, whereas the Academic only guarantees that there will be a "in : cfp.call" in the first cycle, and only a "in : drink.insert_coin" in the second. But this may be by design! The specification of $\mathsf{protspec}(Transaction)$ requires a coin in the first cycle, so it is correct to reject the contract. However, if the designer intended that a coin could be inserted at any point irrespective of the precise moment, she could have written the following alternative protocol specification:

$$
\begin{aligned}
\mathsf{protspec}(Transaction) \stackrel{def}{=} \{\ & [*]\,;\mathsf{insert\_coin}\,; \\
& \{[*]\,;\mathsf{more}\,;[*]\,;\mathsf{insert\_coin}\}\,[*]\,; \\
& [*]\,;\mathsf{change}\,; \\
& \{\{[*]\,;\mathsf{coffee\_button}\,;[*]\,;\mathsf{coffee}\} \mid \{[*]\,;\mathsf{tea\_button}\,;[*]\,;\mathsf{tea}\}\}\quad \}\,[*]
\end{aligned}
$$

Now, the Academic does conform to Transaction.

Algorithm 2 Capsule/interface conformance.

Require: K is a capsule with contract $C = (A, G)$ and interface F with contract C'
Ensure: returns true if C conforms to all port's specifications

```
1: function CHECK-CAPSULE-INTERFACE-CONFORMANCE(K)
2:     F ← interface(K)
3:     C ← contract(K)
4:     C' ← contract(F)
5:     r ← CHECK-CONTRACT-REFINEMENT(C, C')          ▷ Call Algorithm 3
6:     return r
7: end function
```

Algorithm 3 Checking contract refinement.

Require: $C = (A, G)$ and $C' = (A', G')$ are contracts
Ensure: $C \preceq C'$

```
1: function CHECK-CONTRACT-REFINEMENT(C, C')
2:     A ← assumption(C)
3:     G ← guarantee(C)
4:     A' ← assumption(C')
5:     G' ← guarantee(C')
6:     f1 ← (A' → A)
7:     f2 ← (G → (A' → G'))
8:     r1 ← CHECK-VALID-PSL(f1)
9:     if r1 is false then
10:        return false
11:    else
12:        r2 ← CHECK-VALID-PSL(f2)
13:        return r2
14:    end if
15:    return r
16: end function
```

5.2 Conformance between a capsule and its interface

The compatibility between the contract of a capsule and that of an interface can be defined in terms of contract refinement. Intuitively, the capsule's contract must refine the interface contract. This is established whenever the guarantees of the capsule imply the guarantees of the interface, and when the assumptions of the interface imply the assumptions of the capsule.

Answer 8. (To Question 8) Given a capsule K with contract C and interface F, itself with contract C', We say that K conforms to F if $C \preceq C'$. This is done by Algorithm 2 which invokes Algorithm 3 whose correctness is established by Corollary 3 in Section 6.

5.3 Conformance and inheritance

Conformance between a capsule or interface and its parent, is defined in terms of contract refinement, as in Subsection 5.2.

Answer 9. (To Question 9) Given a capsule K with contract C and parent capsule K', itself with contract C', We say that K conforms to K' if $C \preceq C'$. This is done by an algorithm analogous to Algorithm 2 using Algorithm 3.

6 Compositional inference

Theorem 4 in Section 4 established that PSL forms a complete specification theory as defined in Section 3. Hence we can interpret the generic definitions, propositions, lemmas and theorems from the theoretical framework in this context.

Notation 3. We will write $K = K_1 \parallel K_2$ for a composite component K with two sub-components K_1 and K_2. We call such components *binary*. We generalize this notation for a finite number of components and write $K = K_1 \parallel \cdots \parallel K_n$ or $K = \Pi_{i=1}^{n} K_i$ for a composite component K with n sub-components K_i, and call this n-ary composition. We assume that the $\parallel$ operator is commutative and associative so that $K_1 \parallel K_2 = K_2 \parallel K_1$ and $K_1 \parallel (K_2 \parallel K_3) = (K_1 \parallel K_2) \parallel K_3 = K_1 \parallel K_2 \parallel K_3$. Note that this is an informal notation. For the formal definition of composition, see Definition 33.

Proposition 9 (**Relativized refinement in PSL**). *Let $P, Q, R \in$ **CompPSL**. $P \leqslant_{\mathrm{R}}^{\mathrm{psl}} Q$ iff for all R' such that $\vDash R' \to R$, $\vDash P \wedge R' \to Q \wedge R'$.*

Lemma 2. *Let $P, Q, R \in$ **CompPSL**. Then $P \leqslant_{\mathrm{R}}^{\mathrm{psl}} Q$ iff $\vDash P \wedge R \to Q$.*

Proposition 10 (**Contract implementations and environments in PSL**). *Let $C = (A, G)$ be a PSL contract. By*

$$\mathsf{impl}[\![C]\!] = \{I \in \textbf{CompPSL} \mid \vDash I \wedge A \to G\}$$

and

$$\mathsf{env}[\![C]\!] = \{E \in \textbf{CompPSL} \mid \vDash E \to A\}$$

Proposition 11. *Let $C = (A, G)$ and $C' = (A', G')$ be PSL contracts. $C' \preceq C$ if for all implementations $I \in$ **CompPSL**, $\vDash I \wedge A' \to G'$ implies $\vDash I \wedge A \to G$ (see Lemma 2) and for all environments $E \in$ **CompPSL**, $\vDash E \to A$ implies $\vDash E \to A'$.*

Proposition 12. *Let $C = (A, G)$ and $C' = (A', G')$ be PSL contracts. We have that $C \equiv C'$ iff $\vDash A \leftrightarrow A'$ and $\vDash G \leftrightarrow G'$. And $C \simeq C'$ iff (1) for all I, $\vDash I \wedge A \to G$ iff $\vDash I \wedge A' \to G'$, and (2) for all E, $\vDash E \to A$ iff $\vDash E \to A'$.*

Proposition 13. *Let $C = (A, G)$ be a PSL contract. C is in normal form if $\vDash I \wedge A \to G$ iff $\vDash I \to G$.*

Definition 23 (**PSL normal form**). Given a PSL contract $C = (A, G)$, we define $\underline{C} \stackrel{def}{=} (A, \underline{G})$ where $\underline{G} \stackrel{def}{=} A \to G$.

Proposition 14. *Given a PSL contract $C = (A, G)$:*

(i) $C \simeq \underline{C}$

(ii) $\underline{C}$ is in normal form.

This gives as a characterization of valid implementations.

Corollary 2. *For all $I \in$ **CompPSL**, $I \in \mathsf{impl}[\![C]\!]$ if and only if $\vDash I \to (A \to G)$*

Corollary 3. *Let $C = (A, G)$ and $C' = (A', G')$ be to PSL contracts. Then $C' \preceq C$ if and only if $\vDash A \to A'$ and $\vDash G' \to (A \to G)$.*

6.1 Relating formulas and models

Note that in Proposition 10, an "implementation" is actually a **PSL** formula. In practice, an implementation would be a design model which satisfies the formula, or even the code generated from the design model. We will use the term *implementation model* for an actual model which satisfies an implementation specification.

In general it may be possible to transform an implementation model into an implementation specification by using an algorithm similar to the classical algorithm to obtain a regular expression from an NFA or DFA (see, *e.g.*, [Sip97]). Similarly, one can derive a Büchi automaton from a PSL expression (see [CRT08, DLP04]), and presumably, such automaton can be transformed into an $\mathsf{RTEdge}^{\mathrm{TM}}$ atomic capsule. In general such

model is not uniquely defined, but the set of all valid realizations can be seen as an equivalence class, *i.e.*, all realizations of a formula are equivalent in that they satisfy the same formula. Therefore, the transformation just needs to pick a representative element in such equivalence class.

The transformation of an atomic capsule to a formula can be generalized to composite capsules, if we have a way to "flatten" the composite capsule into an equivalent atomic capsule. Assuming that such flattening is available, we can talk about the transformation of a component (atomic or composite) into a formula.

In this report we will not present the details of these transformations, but we will assume these are possible. Henceforth we will use the following definitions for the transformations:

Definition 24 (**Implementation specification and model**)**.** Given an **CompPSL** formula φ, we denote $\mathsf{implmod}[\![\varphi]\!]$ for a (chosen representative) *implementation model* that satisfies φ. Similarly, given an $\mathsf{RTEdge}^{\mathrm{TM}}$ component K, we denote $\mathsf{implspec}[\![K]\!]$ for the *implementation specification* (**PSL** formula) φ that is satisfied by K. These functions $\mathsf{implmod}$ and $\mathsf{implspec}$ must satisfy the following conditions: for any components $K_1, ..., K_n$:

$$\mathsf{implspec}[\![K_1 \parallel \cdots \parallel K_n]\!] = \mathsf{implspec}[\![K_1]\!] \otimes^{\mathsf{psl}} \cdots \otimes^{\mathsf{psl}} \mathsf{implspec}[\![K_n]\!]$$

and for any **CompPSL** formula φ and $\mathsf{RTEdge}^{\mathrm{TM}}$ component K:

$$\mathsf{implspec}[\![K]\!] \leqslant_{\mathsf{psl}} \varphi \text{ if and only if } K \leqslant_{\mathsf{rte}} \mathsf{implmod}[\![\varphi]\!]$$

for a given pair of suitable refinement preorders $\leqslant_{\mathsf{psl}}$ and $\leqslant_{\mathsf{rte}}$ for **CompPSL** and $\mathsf{RTEdge}^{\mathrm{TM}}$ respectively.

The first requirement on $\mathsf{implspec}$ states that it must be an homomorphism for the composition operator, this is, parallel composition of component models must correspond to the composition (conjunction) of their PSL formulas.

The second requirement states that the functions $\mathsf{implmod}$ and $\mathsf{implspec}$ should form a *Galois connection*, this is, that they should be related in such a way that they capture the dual notions of *abstraction* and *realization*. To see this, it is useful to look at the following equivalent characterization of this relation: $\mathsf{implmod}$ and $\mathsf{implspec}$ must satisfy the following four conditions:

(a) if $\varphi_1 \leqslant_{\mathsf{psl}} \varphi_2$ then $\mathsf{implmod}[\![\varphi_1]\!] \leqslant_{\mathsf{rte}} \mathsf{implmod}[\![\varphi_2]\!]$ ($\mathsf{implmod}$ is monotone: if φ_1 is a refinement of φ_2, then the implementation model of φ_1 must be a refinement of φ_2's implementation model)

(b) if $K_1 \leqslant_{\mathsf{rte}} K_2$ then $\mathsf{implspec}[\![K_1]\!] \leqslant_{\mathsf{psl}} \mathsf{implspec}[\![K_2]\!]$ ($\mathsf{implspec}$ is monotone: if component K_1 is a refinement of K_2, then the specification of K_1 must be a refinement of the K_2's specification)

(c) $\mathsf{implspec}[\![\mathsf{implmod}[\![\varphi]\!]]\!] \leqslant_{\mathsf{psl}} \varphi$ ($\mathsf{implspec} \circ \mathsf{implmod}$ is idempotent: the specification of the representative implementation of φ must refine φ)

(d) $K \leqslant_{\mathsf{rte}} \mathsf{implmod}[\![\mathsf{implspec}[\![K]\!]]\!]$ ($\mathsf{implmod} \circ \mathsf{implspec}$ is idempotent: a component K must be a refinement of the representative implementation of K's specification)

Intuitively, these requirements capture the relationship between an abstraction (an implementation specification) and a realization (an implementation model). In our case, the preorder $\leqslant_{\mathsf{psl}}$ can be taken to be the refinement relation for the PSL specification theory (Theorem 4), *i.e.*, $\leqslant_{\mathsf{psl}} = \leqslant^{\mathsf{psl}}$, and the preorder $\leqslant_{\mathsf{rte}}$ can be taken to be trace inclusion, simulation preorder or any other similar relation.

In the rest of this report we assume that suitable functions ($\mathsf{implmod}$ and $\mathsf{implspec}$) and preorder relations ($\leqslant_{\mathsf{psl}}$ and $\leqslant_{\mathsf{rte}}$) have been provided.

Definition 25 (**Capsule correctness**)**.** Given a component K and a PSL contract $C = (A, G)$, we say that K *satisfies* C, written $K \models C$ if $\mathsf{implspec}[\![K]\!] \in \mathsf{impl}[\![C]\!]$. This is, $K \models C$ if $\mathsf{implspec}[\![K]\!] \leqslant_{\mathrm{A}}^{\mathrm{psl}} G$, or in other words, if $\models \mathsf{implspec}[\![K]\!] \wedge A \rightarrow G$, by Proposition 10.

Theorem 6. *Given a component K and two contracts C and C', if $K \models C$ and $C \preceq C'$ then $K \models C'$.*

Algorithm 4 Atomic capsule verification.

Require: K is an atomic capsule annotated with a contract $C = (A, G)$
Ensure: $K \models C$
1: **function** ATOMIC-COMPONENT-VERIF(K,C)
2: $M \leftarrow \mathsf{implspec}[\![K]\!]$ ▷ Transform K into a PSL formula M
3: $R \leftarrow (M \wedge A \rightarrow G)$
4: $r \leftarrow$ CHECK-VALID-PSL(R) ▷ Determine that $M \in \mathsf{impl}[\![C]\!]$
5: **return** r
6: **end function**

6.2 Verifying atomic capsules

Definition 25 gives as a way to verify the correctness of an atomic capsule with respect to a contract.

Answer 10. (To Question 10) Given an atomic capsule K and a PSL contract $C = (A, G)$, we can establish the correctness of K with respect to C, by transforming K into a PSL formula $\mathsf{implspec}[\![K]\!]$ and then verifying if the PSL formula $\mathsf{implspec}[\![K]\!] \wedge A \rightarrow G$ is valid. This is, we can use Algorithm 4.

6.3 Verifying composite capsules: basic compositional inference

The core of the compositional inference step rests on composing contracts, so we apply the concepts from Section 3 to PSL:

Proposition 15. *PSL contract $C = (A, G)$ dominates $C_1 = (A_1, G_1)$ and $C_2 = (A_2, G_2)$ if*

(a) for any I_1 and I_2 such that $\models I_1 \wedge A_1 \rightarrow G_1$ and $\models I_2 \wedge A_2 \rightarrow G_2$ then $\models (I_1 \wedge I_2) \wedge A \rightarrow G$

(b) for any E such that $\models E \rightarrow A$:

- *for any I_1 such that $\models I_1 \wedge A_1 \rightarrow G_1$ then $\models (E \wedge I_1) \rightarrow A_2$, and*
- *for any I_2 such that $\models I_2 \wedge A_2 \rightarrow G_2$ then $\models (E \wedge I_2) \rightarrow A_1$*

Proposition 16. *Given two dominatible PSL contracts $C_1 = (A_1, G_1)$ and $C_2 = (A_2, G_2)$ which have normal forms $\underline{C_1} = (A_1, \underline{G_1})$ and $\underline{C_2} = (A_2, \underline{G_2})$,*

$$C_1 \boxtimes C_2 = (\widetilde{A}, \widetilde{G})$$

where

$$\begin{aligned} \widetilde{A} &= (\underline{G_2} \rightarrow A_1) \wedge (\underline{G_1} \rightarrow A_2) \\ &= ((A_2 \rightarrow G_2) \rightarrow A_1) \wedge ((A_1 \rightarrow G_1) \rightarrow A_2 \end{aligned}$$

and

$$\begin{aligned} \widetilde{G} &= \underline{G_1} \wedge \underline{G_2} \\ &= (A_1 \rightarrow G_1) \wedge (A_2 \rightarrow G_2) \end{aligned}$$

Corollary 4. *Given PSL contracts $C = (A, G)$, $C_1 = (A_1, G_1)$ and $C_2 = (A_2, G_2)$. Then $C_1 \boxtimes C_2 \preceq C$ iff $\models A \rightarrow \widetilde{A}$ and $\models \widetilde{G} \rightarrow \underline{G}$.*

These constructions and the previous results gives us a general approach to the core compositional inference problem, which we can now outline.

Suppose that we have a composite capsule $K = K_1 \parallel K_2$ annotated with a contract C, where the subcapsules K_1 and K_2 are annotated with contracts C_1 and C_2 respectively. An outline of the algorithm to establish if $K \models C$ is as follows:

1. Analyze K_1:

(a) If K_1 is atomic, use Answer 10 to establish $K_1 \vDash C_1$

(b) If K_1 is composite, apply this algorithm recursively

2. Analyze K_2:

(a) If K_2 is atomic, use Answer 10 to establish $K_2 \vDash C_2$

(b) If K_2 is composite, apply this algorithm recursively

3. Construct the contract $C_1 \boxtimes C_2$ as given by Proposition 16

4. Show that $C_1 \boxtimes C_2 \preceq C$ using Corollary 4.

This will establish that $K \vDash C$. This is because by Theorem 2 we know that $C_1 \boxtimes C_2$ dominates both C_1 and C_2 which means that the composition of any implementations of C_1 and C_2 is a valid implementation of $C_1 \boxtimes C_2$, so $K \vDash C_1 \boxtimes C_2$ and since $C_1 \boxtimes C_2 \preceq C$, by Theorem 6 we get $K \vDash C$.

Adapting the formulas

There is, however, a technical problem with this, similar to the one we encountered in Section 5: C_1 and C_2 to not share the same alphabet because they talk about different components, and therefore different ports. Furthermore, the direction of signals in atomic propositions would not match for signals on ports linked by a connector. Consider for example that if K_1 and K_2 are connected, one of the subformulas that we need to prove is $\underline{G_1} \rightarrow A_2$. But $\underline{G_1}$ talks about the ports of K_1 while A_2 talks about the ports of K_2. Hence we need to adapt the formulas in C_1 and C_2 so that they share the same alphabets.

Consider the example in Subsection 1.1, and in particular the Academia1 capsule, depicted in Figure 5 on page 6. In such composite diagrams unfilled ports are base ports, and filled ports are conjugate. So for example, port cfp in the Academic1 capsule is base, but port cfp in the Conference1 capsule is a conjugate port. Let us suppose that we have the following guarantees for sub-components of Academia1:

$$\begin{aligned}
\mathsf{guarantee(Academic1)} &\stackrel{def}{=} \mathsf{G(in : grants.funds \rightarrow F\,out : cfp.paper)} \\
\mathsf{guarantee(Conference1)} &\stackrel{def}{=} \mathsf{G(in : cfp.paper \rightarrow F\,out : papers.paper)} \\
\mathsf{guarantee(Repository1)} &\stackrel{def}{=} \mathsf{G(in : publish.paper \rightarrow F\,out : read.paper)}
\end{aligned}$$

Furthermore, assume that we want to establish the guarantee for the Academia1 composite capsule is:

$$\mathsf{guarantee(Academia1)} \stackrel{def}{=} \mathsf{G(in : grants.funds \rightarrow F\,out : innovation.paper)}$$

Let us also suppose that all assumptions are true, so that

$$\begin{aligned}
\mathsf{contract(Academic1)} &\stackrel{def}{=} (\mathsf{true}, G_1) \\
\mathsf{contract(Conference1)} &\stackrel{def}{=} (\mathsf{true}, G_2) \\
\mathsf{contract(Repository1)} &\stackrel{def}{=} (\mathsf{true}, G_3) \\
\mathsf{contract(Academia1)} &\stackrel{def}{=} (\mathsf{true}, G)
\end{aligned}$$

where $G_1 \stackrel{def}{=} \mathsf{guarantee(Academic1)}$, $G_2 \stackrel{def}{=} \mathsf{guarantee(Conference1)}$, $G_3 \stackrel{def}{=} \mathsf{guarantee(Repository1)}$, and $G \stackrel{def}{=} \mathsf{guarantee(Academia1)}$. Then we have that

$$C_1 \boxtimes C_2 \boxtimes C_3 = (\mathsf{true}, G_1 \wedge G_2 \wedge G_3)$$

where $C_1 \stackrel{def}{=} \mathsf{contract(Academic1)}$, $C_2 \stackrel{def}{=} \mathsf{contract(Conference1)}$ and $C_3 \stackrel{def}{=} \mathsf{contract(Repository1)}$. Then to check that $C_1 \boxtimes C_2 \boxtimes C_3$ refines $C \stackrel{def}{=} \mathsf{contract(Academia1)}$, we will need to establish that $\vDash (G_1 \wedge G_2 \wedge G_3) \rightarrow G$. But then the problem becomes apparent: for example, the signal $\mathsf{out : cfp.paper}$ in G_1 is supposed to be the

same signal as in : cfp.paper in G_2, the signal out : read.paper in G_3 is supposed to be the same signal as out : innovation.paper in G, etc.

Hence, we need to identify the atomic propositions of signals which are meant to be the same. This is done by the following definitions. The main idea behind the adaptation of the contract formulas is to replace the occurrence of port names in atomic propositions with the corresponding connector names, and to adapt the signal directions depending on whether the port in question is base or conjugate. These way, formulas will share the same alphabet.

We define a function ghat which takes a guarantee $G \in$ **GuaraPSL** and two capsules K and K_i: K_i is the capsule that the guarantee talks about and K is its containing capsule. This function returns the PSL formula $\hat{G}$ with each port name p ocurring in any atomic proposition of G renamed with the name of the connector c to which it is hooked up in capsule K, and the direction d of atomic proposition $d : p.s$ is switched from in to out and from out to in if port p is conjugate.

Definition 26 (**Guarantee renaming**)**.** Let the function ghat : **GuaraPSL** → **CompCapsules** → **CompCapsules** → **CompPSL** be defined as shown in Figure 12 on page 40, where we write $\mathsf{ghat}[\![\varphi]\!]_{K,K_i}$ for $\mathsf{ghat}(\varphi)(K)(K_i)$ where $\varphi \in$ **GuaraPSL** is a PSL guarantee specification, $K \in$ **CompCapsules** is the container capsule and $K_i \in \mathsf{parts}(K)$ is the sub-capsule of K that φ talks about.

Using Definition 26, the guarantees in our example are transformed as follows:

$$\begin{aligned}
\mathsf{ghat}[\![G_1]\!]_{\mathsf{Academia1,Academic1}} &\stackrel{def}{=} \mathsf{G}(\mathsf{in} : \mathsf{conn3.funds} \rightarrow \mathsf{F}\,\mathsf{out} : \mathsf{conn2.paper})\\
\mathsf{ghat}[\![G_2]\!]_{\mathsf{Academia1,Conference1}} &\stackrel{def}{=} \mathsf{G}(\mathsf{out} : \mathsf{conn2.paper} \rightarrow \mathsf{F}\,\mathsf{in} : \mathsf{conn4.paper})\\
\mathsf{ghat}[\![G_3]\!]_{\mathsf{Academia1,Repository1}} &\stackrel{def}{=} \mathsf{G}(\mathsf{in} : \mathsf{conn4.paper} \rightarrow \mathsf{F}\,\mathsf{in} : \mathsf{conn5.paper})\\
\mathsf{ghat}[\![G]\!]_{\mathsf{Academia1,Academia1}} &\stackrel{def}{=} \mathsf{G}(\mathsf{in} : \mathsf{conn3.funds} \rightarrow \mathsf{F}\,\mathsf{in} : \mathsf{conn5.paper})
\end{aligned}$$

So for example, in G_1, out : cfp.paper is replaced by out : conn2.paper in $\mathsf{ghat}[\![G_1]\!]_{\mathsf{Academia1,Academic1}}$. Meanwhile, in G_2, in : cfp.paper is replaced by out : conn2.paper in $\mathsf{ghat}[\![G_2]\!]_{\mathsf{Academia1,Conference1}}$ since cfp of Conference1 is conjugate and linked to connector conn2. With this change, the two atomic propositions now refer to the same signal, as expected.

Similarly, we have to adapt the assumption formulas, but these require a slightly different treatment to take int account the notation as explained in Remark 3.

We define a function ahat which takes an assumption $A \in$ **GuaraPSL** and capsules K,K_i with $K_i \in \mathsf{parts}(K)$, and returns the PSL formula $\check{A}$ with each port name p ocurring in any atomic proposition of A renamed with the name of the connector c to which it is hooked up in capsule K, and each $\overline{\mathsf{in}}$ replaced by out and each $\overline{\mathsf{out}}$ replaced by in.

Definition 27 (**Assumption renaming**)**.** Let the function ahat : **AssumPSL** → **CompCapsules** → **CompCapsules** → **CompPSL** be defined as shown in Figure 13 on page 41, where we write $\mathsf{ahat}[\![\varphi]\!]_{K,K_i}$ for $\mathsf{ahat}(\varphi)(K)(K_i)$ where $\varphi \in$ **AssumPSL** is a PSL assumption specification, $K \in$ **CompCapsules** is the container capsule and $K_i \in \mathsf{parts}(K)$ is the sub-capsule of K that φ talks about.

Binary compositional inference

We can now define the proper algorithms. The top-level algorithm is Algorithm 5 which decides whether to apply Algorithm 4 or Algorithm 6 depending on whether it is an atomic capsule or a binary composite capsule. Algorithm 6 in turn invokes Algorithm 7 as the compositional inference step.

The correctness of these algorithms is established by the following.

Theorem 7 (**Correctness of Algorithm 5**)**.** COMPONENT-VERIF$(K, C) =$ true *if and only if* $K \models C$.

Generalizing to n-ary components

To be able to generalize the previous algorithms to composite components with more than two sub-components we need the following:

$$
\begin{array}{rcll}
\mathsf{ghat}[\![\mathsf{in} : p.s(v)]\!]_{K,K_i} & \stackrel{def}{=} & \mathsf{in} : c.s(v) & \text{if } \mathsf{linksmap}(K)(c) = (r.p, x) \\
 & & & \quad \text{or } \mathsf{linksmap}(K)(c) = (x, r.p) \\
 & & & \text{and } \mathsf{kindmap}(\mathsf{interface}(K_i))(p) = \mathsf{base} \\
\mathsf{ghat}[\![\mathsf{in} : p.s(v)]\!]_{K,K_i} & \stackrel{def}{=} & \mathsf{out} : c.s(v) & \text{if } \mathsf{linksmap}(K)(c) = (r.p, x) \\
 & & & \quad \text{or } \mathsf{linksmap}(K)(c) = (x, r.p) \\
 & & & \text{and } \mathsf{kindmap}(\mathsf{interface}(K_i))(p) = \mathsf{conj} \\
\mathsf{ghat}[\![\mathsf{out} : p.s(v)]\!]_{K,K_i} & \stackrel{def}{=} & \mathsf{out} : c.s(v) & \text{if } \mathsf{linksmap}(K)(c) = (r.p, x) \\
 & & & \quad \text{or } \mathsf{linksmap}(K)(c) = (x, r.p) \\
 & & & \text{and } \mathsf{kindmap}(\mathsf{interface}(K_i))(p) = \mathsf{base} \\
\mathsf{ghat}[\![\mathsf{out} : p.s(v)]\!]_{K,K_i} & \stackrel{def}{=} & \mathsf{in} : c.s(v) & \text{if } \mathsf{linksmap}(K)(c) = (r.p, x) \\
 & & & \quad \text{or } \mathsf{linksmap}(K)(c) = (x, r.p) \\
 & & & \text{and } \mathsf{kindmap}(\mathsf{interface}(K_i))(p) = \mathsf{conj} \\
\mathsf{ghat}[\![f(a)]\!]_{K,K_i} & \stackrel{def}{=} & f(a) & \text{if } f(a) \in \mathbf{BoolExpr}_{\mathbf{Attributes}} \\
\mathsf{ghat}[\![!b]\!]_{K,K_i} & \stackrel{def}{=} & !\mathsf{ghat}[\![b]\!]_{K,K_i} & \text{if } b \in \mathbf{BoolExpr} \\
\mathsf{ghat}[\![b_1 \wedge b_2]\!]_{K,K_i} & \stackrel{def}{=} & \mathsf{ghat}[\![b_1]\!]_{K,K_i} \wedge \mathsf{ghat}[\![b_2]\!]_{K,K_i} & \text{if } b_1, b_2 \in \mathbf{BoolExpr} \\
\mathsf{ghat}[\![\{r\}]\!]_{K,K_i} & \stackrel{def}{=} & \{\mathsf{ghat}[\![r]\!]_{K,K_i}\} & \text{if } r \in \mathbf{SERE} \\
\mathsf{ghat}[\![r_1 \,;\, r_2]\!]_{K,K_i} & \stackrel{def}{=} & \{\mathsf{ghat}[\![r_1]\!]_{K,K_i}\} \,;\, \{\mathsf{ghat}[\![r_2]\!]_{K,K_i}\} & \text{if } r_1, r_2 \in \mathbf{SERE} \\
\mathsf{ghat}[\![r_1 : r_2]\!]_{K,K_i} & \stackrel{def}{=} & \{\mathsf{ghat}[\![r_1]\!]_{K,K_i}\} : \{\mathsf{ghat}[\![r_2]\!]_{K,K_i}\} & \text{if } r_1, r_2 \in \mathbf{SERE} \\
\mathsf{ghat}[\![r_1 \mid r_2]\!]_{K,K_i} & \stackrel{def}{=} & \{\mathsf{ghat}[\![r_1]\!]_{K,K_i}\} \mid \{\mathsf{ghat}[\![r_2]\!]_{K,K_i}\} & \text{if } r_1, r_2 \in \mathbf{SERE} \\
\mathsf{ghat}[\![r_1 \,\&\&\, r_2]\!]_{K,K_i} & \stackrel{def}{=} & \{\mathsf{ghat}[\![r_1]\!]_{K,K_i}\} \,\&\&\, \{\mathsf{ghat}[\![r_2]\!]_{K,K_i}\} & \text{if } r_1, r_2 \in \mathbf{SERE} \\
\mathsf{ghat}[\![r\,[*0]]\!]_{K,K_i} & \stackrel{def}{=} & \{\mathsf{ghat}[\![r]\!]_{K,K_i}\}\,[*0] & \text{if } r \in \mathbf{SERE} \\
\mathsf{ghat}[\![r\,[*]]\!]_{K,K_i} & \stackrel{def}{=} & \{\mathsf{ghat}[\![r]\!]_{K,K_i}\}\,[*] & \text{if } r \in \mathbf{SERE} \\
\mathsf{ghat}[\![b!]\!]_{K,K_i} & \stackrel{def}{=} & \mathsf{ghat}[\![b]\!]_{K,K_i}! & \text{if } b \in \mathbf{BoolExpr} \\
\mathsf{ghat}[\![(\varphi)]\!]_{K,K_i} & \stackrel{def}{=} & (\mathsf{ghat}[\![\varphi]\!]_{K,K_i}) & \text{if } \varphi \in \mathbf{FL} \\
\mathsf{ghat}[\![\neg\varphi]\!]_{K,K_i} & \stackrel{def}{=} & \neg\mathsf{ghat}[\![\varphi]\!]_{K,K_i} & \text{if } \varphi \in \mathbf{FL} \\
\mathsf{ghat}[\![\varphi_1 \wedge \varphi_2]\!]_{K,K_i} & \stackrel{def}{=} & (\mathsf{ghat}[\![\varphi_1]\!]_{K,K_i}) \wedge (\mathsf{ghat}[\![\varphi_2]\!]_{K,K_i}) & \text{if } \varphi_1, \varphi_2 \in \mathbf{FL} \\
\mathsf{ghat}[\![\mathsf{X!}\,\varphi]\!]_{K,K_i} & \stackrel{def}{=} & \mathsf{X!}(\mathsf{ghat}[\![\varphi]\!]_{K,K_i}) & \text{if } \varphi \in \mathbf{FL} \\
\mathsf{ghat}[\![[\varphi_1 \,\mathsf{U}\, \varphi_2]]\!]_{K,K_i} & \stackrel{def}{=} & [(\mathsf{ghat}[\![\varphi_1]\!]_{K,K_i})\,\mathsf{U}(\mathsf{ghat}[\![\varphi_1]\!]_{K,K_i})] & \text{if } \varphi_1, \varphi_2 \in \mathbf{FL} \\
\mathsf{ghat}[\![\varphi\,\mathsf{abort}\,b]\!]_{K,K_i} & \stackrel{def}{=} & (\mathsf{ghat}[\![\varphi]\!]_{K,K_i})\,\mathsf{abort}(\mathsf{ghat}[\![b]\!]_{K,K_i}) & \text{if } \varphi \in \mathbf{FL}, b \in \mathbf{BoolExpr} \\
\mathsf{ghat}[\![r \mapsto \varphi]\!]_{K,K_i} & \stackrel{def}{=} & (\mathsf{ghat}[\![r]\!]_{K,K_i}) \mapsto (\mathsf{ghat}[\![\varphi]\!]_{K,K_i}) & \text{if } \varphi \in \mathbf{FL}, r \in \mathbf{SERE} \\
\mathsf{ghat}[\![r!]\!]_{K,K_i} & \stackrel{def}{=} & \{\mathsf{ghat}[\![r]\!]_{K,K_i}\}! & \text{if } r \in \mathbf{SERE}
\end{array}
$$

Figure 12: Guarantee renaming.

$$
\begin{array}{rcll}
\mathsf{ahat}[\![\overline{\mathsf{in}} : p.s(v)]\!]_{K,K_i} & \stackrel{def}{=} & \mathsf{out} : c.s(v) & \text{if } \mathsf{linksmap}(K)(c) = (r.p, x) \\
 & & & \quad \text{or } \mathsf{linksmap}(K)(c) = (x, r.p) \\
 & & & \text{and } \mathsf{kindmap}(\mathsf{interface}(K_i))(p) = \mathsf{base} \\
\mathsf{ahat}[\![\overline{\mathsf{in}} : p.s(v)]\!]_{K,K_i} & \stackrel{def}{=} & \mathsf{in} : c.s(v) & \text{if } \mathsf{linksmap}(K)(c) = (r.p, x) \\
 & & & \quad \text{or } \mathsf{linksmap}(K)(c) = (x, r.p) \\
 & & & \text{and } \mathsf{kindmap}(\mathsf{interface}(K_i))(p) = \mathsf{conj} \\
\mathsf{ahat}[\![\overline{\mathsf{out}} : p.s(v)]\!]_{K,K_i} & \stackrel{def}{=} & \mathsf{in} : c.s(v) & \text{if } \mathsf{linksmap}(K)(c) = (r.p, x) \\
 & & & \quad \text{or } \mathsf{linksmap}(K)(c) = (x, r.p) \\
 & & & \text{and } \mathsf{kindmap}(\mathsf{interface}(K_i))(p) = \mathsf{base} \\
\mathsf{ahat}[\![\overline{\mathsf{out}} : p.s(v)]\!]_{K,K_i} & \stackrel{def}{=} & \mathsf{out} : c.s(v) & \text{if } \mathsf{linksmap}(K)(c) = (r.p, x) \\
 & & & \quad \text{or } \mathsf{linksmap}(K)(c) = (x, r.p) \\
 & & & \text{and } \mathsf{kindmap}(\mathsf{interface}(K_i))(p) = \mathsf{conj} \\
\mathsf{ahat}[\![f(a)]\!]_{K,K_i} & \stackrel{def}{=} & f(a) & \text{if } f(a) \in \mathbf{BoolExpr}_{\mathbf{Attributes}} \\
\mathsf{ahat}[\![!b]\!]_{K,K_i} & \stackrel{def}{=} & !\mathsf{ahat}[\![b]\!]_{K,K_i} & \text{if } b \in \mathbf{BoolExpr} \\
\mathsf{ahat}[\![b_1 \wedge b_2]\!]_{K,K_i} & \stackrel{def}{=} & \mathsf{ahat}[\![b_1]\!]_{K,K_i} \wedge \mathsf{ahat}[\![b_2]\!]_{K,K_i} & \text{if } b_1, b_2 \in \mathbf{BoolExpr} \\
\mathsf{ahat}[\![\{r\}]\!]_{K,K_i} & \stackrel{def}{=} & \{\mathsf{ahat}[\![r]\!]_{K,K_i}\} & \text{if } r \in \mathbf{SERE} \\
\mathsf{ahat}[\![r_1 \,;\, r_2]\!]_{K,K_i} & \stackrel{def}{=} & \{\mathsf{ahat}[\![r_1]\!]_{K,K_i}\} \,;\, \{\mathsf{ahat}[\![r_2]\!]_{K,K_i}\} & \text{if } r_1, r_2 \in \mathbf{SERE} \\
\mathsf{ahat}[\![r_1 : r_2]\!]_{K,K_i} & \stackrel{def}{=} & \{\mathsf{ahat}[\![r_1]\!]_{K,K_i}\} : \{\mathsf{ahat}[\![r_2]\!]_{K,K_i}\} & \text{if } r_1, r_2 \in \mathbf{SERE} \\
\mathsf{ahat}[\![r_1 \mid r_2]\!]_{K,K_i} & \stackrel{def}{=} & \{\mathsf{ahat}[\![r_1]\!]_{K,K_i}\} \mid \{\mathsf{ahat}[\![r_2]\!]_{K,K_i}\} & \text{if } r_1, r_2 \in \mathbf{SERE} \\
\mathsf{ahat}[\![r_1 \,\&\&\, r_2]\!]_{K,K_i} & \stackrel{def}{=} & \{\mathsf{ahat}[\![r_1]\!]_{K,K_i}\} \,\&\&\, \{\mathsf{ahat}[\![r_2]\!]_{K,K_i}\} & \text{if } r_1, r_2 \in \mathbf{SERE} \\
\mathsf{ahat}[\![r\,[*0]]\!]_{K,K_i} & \stackrel{def}{=} & \{\mathsf{ahat}[\![r]\!]_{K,K_i}\}\,[*0] & \text{if } r \in \mathbf{SERE} \\
\mathsf{ahat}[\![r\,[*]]\!]_{K,K} & \stackrel{def}{=} & \{\mathsf{ahat}[\![r]\!]_{K,K_i}\}\,[*] & \text{if } r \in \mathbf{SERE} \\
\mathsf{ahat}[\![b!]\!]_{K,K_i} & \stackrel{def}{=} & \mathsf{ahat}[\![b]\!]_{K,K_i}! & \text{if } b \in \mathbf{BoolExpr} \\
\mathsf{ahat}[\![(\varphi)]\!]_{K,K_i} & \stackrel{def}{=} & (\mathsf{ahat}[\![\varphi]\!]_{K,K_i}) & \text{if } \varphi \in \mathbf{FL} \\
\mathsf{ahat}[\![\neg\varphi]\!]_{K,K_i} & \stackrel{def}{=} & \neg\mathsf{ahat}[\![\varphi]\!]_{K,K_i} & \text{if } \varphi \in \mathbf{FL} \\
\mathsf{ahat}[\![\varphi_1 \wedge \varphi_2]\!]_{K,K_i} & \stackrel{def}{=} & (\mathsf{ahat}[\![\varphi_1]\!]_{K,K_i}) \wedge (\mathsf{ahat}[\![\varphi_2]\!]_{K,K_i}) & \text{if } \varphi_1, \varphi_2 \in \mathbf{FL} \\
\mathsf{ahat}[\![\mathsf{X}!\,\varphi]\!]_{K,K_i} & \stackrel{def}{=} & \mathsf{X}!(\mathsf{ahat}[\![\varphi]\!]_{K,K_i}) & \text{if } \varphi \in \mathbf{FL} \\
\mathsf{ahat}[\![[\varphi_1 \,\mathsf{U}\, \varphi_2]]\!]_{K,K_i} & \stackrel{def}{=} & [(\mathsf{ahat}[\![\varphi_1]\!]_{K,K_i}) \,\mathsf{U}(\mathsf{ahat}[\![\varphi_1]\!]_{K,K_i})] & \text{if } \varphi_1, \varphi_2 \in \mathbf{FL} \\
\mathsf{ahat}[\![\varphi \,\mathsf{abort}\, b]\!]_{K,K_i} & \stackrel{def}{=} & (\mathsf{ahat}[\![\varphi]\!]_{K,K_i}) \,\mathsf{abort}(\mathsf{ahat}[\![b]\!]_{K,K_i}) & \text{if } \varphi \in \mathbf{FL}, b \in \mathbf{BoolExpr} \\
\mathsf{ahat}[\![r \mapsto \varphi]\!]_{K,K_i} & \stackrel{def}{=} & (\mathsf{ahat}[\![r]\!]_{K,K_i}) \mapsto (\mathsf{ahat}[\![\varphi]\!]_{K,K_i}) & \text{if } \varphi \in \mathbf{FL}, r \in \mathbf{SERE} \\
\mathsf{ahat}[\![r!]\!]_{K,K_i} & \stackrel{def}{=} & \{\mathsf{ahat}[\![r]\!]_{K,K_i}\}! & \text{if } r \in \mathbf{SERE}
\end{array}
$$

Figure 13: Assumption renaming.

Algorithm 5 Basic capsule verification.

Require: K is either an atomic capsule annotated with a contract $C = (A, G)$ or a composite capsule $K = K_1 \parallel K_2$ annotated with a PSL contract $C = (A, G)$, and sub-components K_1 and K_2 annotated with PSL contracts $C_1 = (A_1, G_1)$ and $C_2 = (A_2, G_2)$

Ensure: $K \models C$

function COMPONENT-VERIF(K,C)
  if K is an atomic capsule **then**
    return ATOMIC-COMPONENT-VERIF(K,C) ▷ Call Algorithm 4
  else
    $\{K_1, K_2\} \leftarrow \mathsf{parts}(K)$
    $\{C_1, C_2\} \leftarrow \{\, \mathsf{contract}(K_1), \mathsf{contract}(K_2) \,\}$
    return BINARY-COMPONENT-ANALYSIS(K,C,$\{K_1, K_2\}$,$\{C_1, C_2\}$) ▷ Call Algorithm 6
  end if
end function

Algorithm 6 Binary compositional analysis.

Require: A composite capsule $K = K_1 \parallel K_2$ annotated with a PSL contract $C = (A, G)$, and sub-components K_1 and K_2 annotated with PSL contracts $C_1 = (A_1, G_1)$ and $C_2 = (A_2, G_2)$

Ensure: $K \models C$

function BINARY-COMPONENT-ANALYSIS(K,C, $\{K_1, K_2\}$,$\{C_1, C_2\}$)
  $v_1 \leftarrow$ COMPONENT-VERIF(K_1,C_1) ▷ Call Algorithm 5
  $v_2 \leftarrow$ COMPONENT-VERIF(K_2,C_2) ▷ Call Algorithm 5
  $u \leftarrow$ BIN-CONTRACT-INFERENCE(K,C,$\{K_1, K_2\}$,$\{C_1, C_2\}$) ▷ Call Algorithm 7
  if v_1 and v_2 and u **then**
    return true
  else
    return false
  end if
end function

Algorithm 7 Binary compositional inference.

Require: $C = (A, G)$, $C_1 = (A_1, G_1)$ and $C_2 = (A_2, G_2)$ must be PSL contracts, K is a composite component annotated with C

Ensure: $C_1 \boxtimes C_2 \preceq C$

function BINARY-CONTRACT-INFERENCE(K,C,$\{K_1, K_2\}$,$\{C_1, C_2\}$)
  $A \leftarrow \mathsf{assumption}(C)$
  $G \leftarrow \mathsf{guarantee}(C)$
  $G^{\mathsf{nf}} \leftarrow (A \rightarrow G)$
  $A_1 \leftarrow \mathsf{assumption}(C_1)$
  $G_1 \leftarrow \mathsf{guarantee}(C_1)$
  $G_1^{\mathsf{nf}} \leftarrow (A_1 \rightarrow G_1)$
  $A_2 \leftarrow \mathsf{assumption}(C_2)$
  $G_2 \leftarrow \mathsf{guarantee}(C_2)$
  $G_2^{\mathsf{nf}} \leftarrow (A_1 \rightarrow G_2)$
  $C_1' \leftarrow (\mathsf{ahat}[\![A_1]\!]_{K,K_1}, \mathsf{ghat}[\![G_1^{\mathsf{nf}}]\!]_{K,K_1})$
  $C_2' \leftarrow (\mathsf{ahat}[\![A_2]\!]_{K,K_2}, \mathsf{ghat}[\![G_2^{\mathsf{nf}}]\!]_{K,K_2})$
  $C' \leftarrow$ BINARY-CONTRACT-COMPOSITION(C_1', C_2') ▷ Call Algorithm 8
  $C'' \leftarrow (\mathsf{ahat}[\![A]\!]_{K,K}, \mathsf{ghat}[\![G^{\mathsf{nf}}]\!]_{K,K})$
  $r \leftarrow$ CHECK-CONTRACT-REFINEMENT(C', C'') ▷ Call Algorithm 3
  return r
end function

Algorithm 8 Binary contract composition.

Require: $C_1 = (A_1, G_1)$ and $C_2 = (A_2, G_2)$ must be PSL contracts
Ensure: returns $C_1 \boxtimes C_2$

function BINARY-CONTRACT-COMPOSITION(C_1,C_2)
  $A_1 \leftarrow \mathsf{assumption}(C_1)$
  $G_1 \leftarrow \mathsf{guarantee}(C_1)$
  $A_2 \leftarrow \mathsf{assumption}(C_2)$
  $G_2 \leftarrow \mathsf{guarantee}(C_2)$
  $G_1^{\mathsf{nf}} \leftarrow (A_1 \rightarrow G_1)$
  $G_2^{\mathsf{nf}} \leftarrow (A_2 \rightarrow G_2)$
  $\widetilde{A} \leftarrow (G_1^{\mathsf{nf}} \rightarrow A_2) \wedge (G_2^{\mathsf{nf}} \rightarrow A_1)$
  $\widetilde{G} \leftarrow (G_1^{\mathsf{nf}} \wedge G_2^{\mathsf{nf}})$
  return $(\widetilde{A}, \widetilde{G})$
end function

Proposition 17 (**Contract composition is commutative and associative and preserves strong contract equivalence**). *For any contracts C_1, C_2, C_3 in a simplified specification theory:*

(i) $C_1 \boxtimes C_2 \equiv C_2 \boxtimes C_1$

(ii) $(C_1 \boxtimes C_2) \boxtimes C_3 \simeq C_1 \boxtimes (C_2 \boxtimes C_3)$

(iii) if $C_1 \equiv C_2$ and C_1 and C_2 are in normal form, then $C_1 \boxtimes C \equiv C_2 \boxtimes C$

This proposition allows us to define the composition of an arbitrary number of contracts:

Definition 28 (n-**ary contract composition**). Given contracts $C_1, ..., C_n$ we define

$$\boxtimes_{i=1}^{n} C_i \stackrel{def}{=} \begin{cases} C_1 & \text{if } n = 1 \\ (\boxtimes_{i=1}^{n-1} C_i) \boxtimes C_n & \text{if } n > 1 \end{cases}$$

Proposition 18. *If $n > 1$ then $\boxtimes_{i=1}^{n} C_i \equiv C_1 \boxtimes (\boxtimes_{i=2}^{n} C_i)$*

Hence the parenthesis are superfluous in an n-ary compostion, which means that we can write

$$\boxtimes_{i=1}^{n} C_i = C_1 \boxtimes \cdots \boxtimes C_n$$

This n-ary composition can be characterized as follows:

Proposition 19. *Let $I = \{1, ..., n\}$ and $\{C_i\}_{i \in I}$ a family of contracts. Then $\boxtimes_{i \in I} C_i = (\widetilde{A}, \widetilde{G})$ where*

$$\widetilde{A} = \bigwedge_{i \in I} (A_i / (\bigwedge_{j \in I \setminus \{i\}} G_i))$$

and

$$\widetilde{G} = \bigwedge_{i \in I} G_i$$

Since we can define n-ary composition, we can generalize the binary algorithms with Algorithm 9, Algorithm 10 and Algorithm 11.

Finally, we can provide an answer to Question 12 and Question 11.

Answer 11. (To Question 12) We can establish that a composite capsule K satisfies its contract C if we already know that each of its sub-components K_1, ..., K_n satisfy their respective contracts C_1, ..., C_n by constructing the contract composition $\boxtimes_{i=1}^{n} C_i$ and checking that it is a refinement of C. This is what Algorithm 11 does.

Answer 12. (To Question 11) We establish that a composite capsule satisfies its contract by recursively verifying the sub-capsules and then performing compositional inference as described in Algorithm 10.

Algorithm 9 General capsule verification.

Require: K is either an atomic capsule annotated with a contract $C = (A, G)$ or a composite capsule $K = \Pi_{i=1}^{n} K_i$ annotated with a PSL contract $C = (A, G)$, and sub-components K_i annotated with PSL contracts $C_i = (A_i, G_i)$ for $i \in \{1, ..., n\}$

Ensure: $K \models C$

function GENERAL-COMPONENT-VERIF(K,C)
  if K is an atomic capsule **then**
    return ATOMIC-COMPONENT-VERIF(K,C) ▷ Call Algorithm 4
  else
    $\{K_1, ..., K_n\} \leftarrow \mathsf{parts}(K)$
    $\{C_1, ..., C_n\} \leftarrow \{\, \mathsf{contract}(K_i) \mid K_i \in \{K_1, ..., K_n\} \,\}$
    return N-ARY-COMPONENT-ANALYSIS(K,C,$\{K_1, ..., K_n\}$,$\{C_1, ..., C_n\}$) ▷ Call Algorithm 10
  end if
end function

Algorithm 10 n-ary compositional analysis.

Require: A composite capsule $K = \Pi_{i=1}^{n} K_i$ annotated with a PSL contract $C = (A, G)$, and sub-components K_i annotated with PSL contracts $C_i = (A_i, G_i)$ for $i \in \{1, ..., n\}$

Ensure: $K \models C$

function N-ARY-COMPONENT-ANALYSIS(K,C,$\{K_1, ..., K_n\}$,$\{C_1, ..., C_n\}$)
  for $i \in \{1, ..., n\}$ **do**
    $v_i \leftarrow$ GENERAL-COMPONENT-VERIF(K_i,C_i) ▷ Call Algorithm 9
    if v_i is false **then**
      return false
    end if
  end for
  $u \leftarrow$ N-ARY-CONTRACT-INFERENCE(K,C,$\{K_1, ..., K_n\}$,$\{C_1, .., C_n\}$) ▷ Call Algorithm 11
  if u is true **then**
    return true
  else
    return false
  end if
end function

Algorithm 11 n-ary compositional inference.

Require: $C = (A, G)$, $C_i = (A_i, G_i)$ for $i \in \{1, ..., n\}$ must be PSL contracts, K is a composite component annotated with C

Ensure: $\boxtimes_{i=1}^{n} C_i \preceq C$

function N-ARY-CONTRACT-INFERENCE(K,C,$\{K_1, ..., K_n\}$,$\{C_1, ..., C_n\}$)
  $A \leftarrow \mathsf{assumption}(C)$
  $G \leftarrow \mathsf{guarantee}(C)$
  $G^{\mathsf{nf}} \leftarrow (A \rightarrow G)$
  for $i \in \{1, ..., n\}$ **do**
    $A_i \leftarrow \mathsf{assumption}(C_i)$
    $G_i \leftarrow \mathsf{guarantee}(C_i)$
    $G_i^{\mathsf{nf}} \leftarrow (A_i \rightarrow G_i)$
    $C_i' \leftarrow (\mathsf{ahat}[\![A_i]\!]_{K,K_i}, \mathsf{ghat}[\![G_i^{\mathsf{nf}}]\!]_{K,K_i})$
  end for
  $C' \leftarrow$ N-ARY-CONTRACT-COMPOSITION($\{C_1', ..., C_n'\}$) ▷ Call Algorithm 12
  $C'' \leftarrow (\mathsf{ahat}[\![A]\!]_{K,K}, \mathsf{ghat}[\![G^{\mathsf{nf}}]\!]_{K,K})$
  $r \leftarrow$ CHECK-CONTRACT-REFINEMENT(C', C'') ▷ Call Algorithm 3
end function

Algorithm 12 n-ary contract composition.

Require: $C_1 = (A_1, G_1)$, ..., $C_n = (A_n, G_n)$ must be PSL contracts
Ensure: returns $\boxtimes_{i=1}^{n} C_i$
1: **function** N-ARY-CONTRACT-COMPOSITION($\{C_1, ..., C_n\}$)
2: $I \leftarrow \{1, ..., n\}$
3: $\widetilde{A} \leftarrow \bigwedge_{i \in I}((\bigwedge_{j \in I \setminus \{i\}} \underline{G_j}) \rightarrow A_i)$
4: $\widetilde{G} \leftarrow \bigwedge_{i \in I} \underline{G_i}$
5: **return** $(\widetilde{A}, \widetilde{G})$
6: **end function**

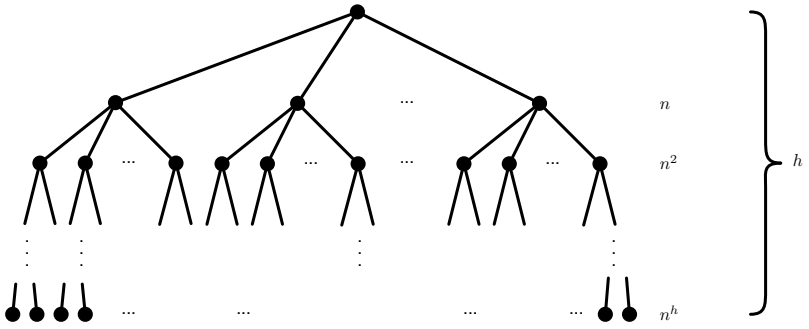

Figure 14: Maximum tree structure of a composite capsule with height h and n sub-capsules in each composite capsule.

Complexity

All algorithms introduced depend ultimately on the CHECK-VALID-PSL function, so their complexity depends directly on the complexity of checking for validity of PSL formulas. Since LTL is a (strict) subset of PSL, verifying PSL is at least as difficult as verifying LTL. In [SC85] it was established that the decision problem for LTL (including past operators) is in PSPACE, so it can be solved using a polynomial amount of space (in terms of the size of the formula), but PSPACE contains the class NP, which means that there is no known worst-case polynomial time solution. Hence the complexity of the PSL validity problem is, as far as is currently known, $O(2^{|\varphi|})$, *i.e.*, exponential in the size $|\varphi|$ of the formula φ.

The heart of the compositional inference mechanism consists on checking the refinement $\boxtimes_{i=1}^{n} C_i \preceq C$, and therefore the complexity depends on the size of $\widetilde{A} = \mathsf{assumption}(\boxtimes_{i=1}^{n} C_i)$ and $\widetilde{G} = \mathsf{guarantee}(\boxtimes_{i=1}^{n} C_i)$. But by Proposition 19, $|\widetilde{A}| \leqslant kn(n-1)$ and $|\widetilde{G}| = k'n$ for some constants k and k', and where n is the number of sub-capsules of the composite capsule being analyzed. Hence, the complexity for the inference algorithm is $O(2^{kn(n-1)})$.

Suppose we have a composite capsule with the tree structure depicted in Figure 14 on page 45, *i.e.*, with n immediate sub-components, each of which also has at most n sub-components and so on, with a maximum height (depth) of h. If we perform the full analysis on such capsule, then, since the complexity for each internal node is $O(2^{kn(n-1)})$, and there are $m = \sum_{j=0}^{h-1} n^j = {}^{n^h-1}\!/_{n-1}$ internal nodes, then the total complexity will be $O(m2^{kn(n-1)} + sn^h) = O(({}^{n^h-1}\!/_{n-1})2^{kn(n-1)} + sn^h) = O(({}^{l-1}\!/_{n-1})2^{kn(n-1)} + sl)$ where $l = n^h$ is the maximum number of all atomic capsules (the leaves of the tree) and s is the maximum number of states for each atomic capsule. If we restrict ourselves to one level of nesting ($h = 1$) then the complexity is $O(2^{kn(n-1)} + sn)$.

In practice however, the number of internal nodes is almost always smaller than the maximum m, and more importantly, the complexity is exponential on the number of components n and not on the number of states. This compares favourably to a monolithic analysis which would flatten the structure of the tree (by a synchronous product or equivalent construction) resulting in s^l total states (with $l = n^h$), and therefore a complexity of $O(2^{s^l})$. If we restrict ourselves to one level of nesting ($h = 1$) then the complexity with flattening is $O(2^{s^n})$. Furthermore, this is the complexity of the full analysis on the whole model, but as

Algorithm 13 Incremental analysis: component changes, but not contract.

Require: K is a composite capsule $K = \Pi_{i=1}^{n} K_i$ annotated with a PSL contract $C = (A, G)$, and subcomponents K_1 and K_2 annotated with PSL contracts $C_1 = (A_1, G_1)$, ..., $C_n = (A_n, G_n)$; COMPONENT-VERIF(K,C) returned true. K_i' is a capsule intended to replace K_i

Ensure: $K \models C$

```
1: function INC-COMPONENT-CHANGE-VERIF(K,C,i,K_i')
2:     {K_1, ..., K_n} ← parts(K)
3:     C_i ← contract(K_i)
4:     r ←GENERAL-COMPONENT-VERIF(K_i',C_i)          ▷ Call Algorithm 9
5:     return r
6: end function
```

described in Section 7, compositional inference enables incremental analysis, where one needs to run the verification only on components which have changed and then run the compositional inference algorithm.

7 Incremental analysis

Once we have the compositional analysis framework in place, we are able to perform *incremental analysis*, this is, analyzing a system during the development process by reusing previous analysis results and combine them with the results of analysing only the parts of the model which have changed.

Answer 13. (To Question 13) Assume that we have a design model K with subcomponents $K_1, ..., K_n$. Furthermore, assume that they are annotated with contracts C, C_1, ..., C_n respectively and that we have already performed compositional analysis with Algorithm 9. Now, suppose that the model changes. There are several possible changes that can be made on K:

(a) A sub-component K_i may be changed (but its contract C_i remains unchanged)

(b) A sub-component K_{n+1} may be added (with a contract C_{n+1})

(c) A sub-component K_i may be removed

(d) The contract C_i of a sub-component C_i may be changed

(e) The contract C of the composite K may be changed

(f) A combination of the above

We consider the first five as the basic modification mechanisms. Each of these operations can be handled as follows:

(a) If sub-component K_i changed and its contract C_i remains unchanged, then we only need to verify $K_i \models C_i$, and if it holds, then $K \models C$, *i.e.*, the composite will still satisfy its contract. If it fails, $K \not\models C$ and some changes must be made by the user, either by modifying K_i or C_i and possibly other components, and then re-running the analysis. See Algorithm 13.

(b) Since we already performed analysis on the first n components, we only need to verify $K_{n+1} \models C_{n+1}$, compose C_{n+1} with the other conctracts and perform the compositional inference step. See Algorithm 14.

(c) When a component is removed, we can simply invoke Algorithm 11 with only the remaining contracts. This will construct the contract composition of the remaining contracts, so the contract from the removed component will not be assumed in the analysis.

(d) If the contract C_i of a sub-component K_i changes to a new contract C_i', then we need to both check that $K_i \models C_i'$ and perform the compositional inference. See Algorithm 15.

Algorithm 14 Incremental analysis: adding a new component.

Require: K is a composite capsule $K = \Pi_{i=1}^{n} K_i$ annotated with a PSL contract $C = (A, G)$, and sub-components $K_1, ..., K_n$ annotated with PSL contracts $C_1 = (A_1, G_1)$, ..., $C_n = (A_n, G_n)$; COMPONENT-VERIF(K,C) returned true. K_{n+1} is a new capsule annotated with contract C_{n+1}

Ensure: $K \models C$

```
1: function INC-COMPONENT-ADD-VERIF(K,C,K_{n+1},C_{n+1})
2:     r ←GENERAL-COMPONENT-VERIF(K_{n+1},C_{n+1})                ▷ Call Algorithm 9
3:     if r is false then
4:         return false
5:     else
6:         c ←N-ARY-CONTRACT-INFERENCE(K,C,{K_1, ..., K_n},{C_1, .., C_n, C_{n+1}})   ▷ Call Algorithm 11
7:         return c
8:     end if
9: end function
```

Algorithm 15 Incremental analysis: changing a sub-contract.

Require: K is a composite capsule $K = \Pi_{i=1}^{n} K_i$ annotated with a PSL contract $C = (A, G)$, and sub-components $K_1, ..., K_n$ annotated with PSL contracts $C_1 = (A_1, G_1)$, ..., $C_n = (A_n, G_n)$; COMPONENT-VERIF(K,C) returned true. C_i' is a new contract replacing C_i

Ensure: $K \models C$

```
1: function INC-CONTRACT-CHANGE-VERIF(K,C,i,C_i')
2:     {K_1, ..., K_n} ← parts(K)
3:     r ←COMPONENT-VERIF(K_i,C_i')                               ▷ Call Algorithm 9
4:     if r is false then
5:         return false
6:     else
7:         {C_1, ..., C_i, ..., C_n} ← { contract(K_i) | K_i ∈ {K_1, ..., K_n} }
8:         L ← {C_1, ..., C_i', ..., C_n}
9:         c ←N-ARY-CONTRACT-INFERENCE(K,C,{K_1, ..., K_n},L)     ▷ Call Algorithm 11
10:        return c
11:    end if
12: end function
```

(e) If only the top-level contract changes, we only need to perform contract inference invoking Algorithm 11 with the new contract.

The algorithms presented here are pessimistic in the sense that if a sub-component K_i changes, the full analysis is performed on K_i by invoking Algorithm 9. This however, may be wasteful, in case that K_i is a composite capsule and the change to K_i was minor. In fact, the change to K_i could be one of the modifications listed above, and thus, the verification of K_i itself could be done by incremental analysis. could be optimized further. We leave this optimization as future work.

8 Quotienting

Recall that the quotienting or "missing part" problem is the following (Question 14):

Given a composite capsule K with contract C and sub-capsules $K_1, ..., K_n$ with contracts $C_1, ..., C_n$ and a sub-capsule placeholder X:

1. What contract C_X should X have so that if we put, in place of X, a component implementation K_X that satisfies C_X and each K_i satisfies C_i then we can conclude that K satisfies C?

2. What should be an implementation of X that satisfies such contract C_X?

If we restrict ourselves to binary components, we kave a component $K = K_1 \parallel X$ with a contract C where K_1 is a component with contract C_1 and X is a component placeholder for which we have no contract. The first question is to find a contract C_X such that $C_1 \boxtimes C_X \preceq C$. If we find such a contract, and we find an implementation K_X of X that satisfies C_X, *i.e.*, $K_X \vDash C_X$, then $K' = K[X \mapsto K_X] = K_1 \parallel K_X$ will satisfy C: $K' \vDash C$ by Theorem 6.

Since we are looking for a C_X such that $C_1 \boxtimes C_X \preceq C$, and in fact we are looking for the weakest such C_X, we call it the quotient of contracts C and C_1 and denote it $C_X = C/C_1$.

Definition 29 (**Contract quotien**t)**.** Given a pair of contracts $C_0 = (A_0, G_0)$ and $C_1 = (A_1, G_1)$, define

$$C_0/C_1 \stackrel{def}{=} (\ddot{A}, \ddot{G})$$

where

$$\ddot{A} \stackrel{def}{=} G_1 \otimes A_0$$

and

$$\ddot{G} \stackrel{def}{=} (G_0/G_1) \wedge (A_1/A_0)$$

Theorem 8. *Given a pair of contracts $C_0 = (A_0, G_0)$ and $C_1 = (A_1, G_1)$, if C_0 and C_1 are in normal form, then $C_1 \boxtimes (C_0/C_1) \preceq C_0$.*

9 Implementing verification

In order to implement the algorithms described in this report we need two things:

1. An algorithm that implements the implspec function to transform atomic capsules into PSL formulas, and
2. An algorithm that for a given PSL formula φ, checks $\models \varphi$, *i.e.*, the validity of the formula, this is, an algorithm that implements the CHECK-VALID-PSL function used in our algorithms.

The first algorithm will be custom-made for RTEdge™ atomic capsules, but it can be based on general algorithms that transform NFAs or DFAs into regular expressions which can be found in any automata theory textbook (*e.g.*, [Sip97]). The regular expressions can be transformed into PSL SEREs.

The second algorithm can be implemented by either a custom-built PSL verifier or an off-the-shelf tool.

9.1 PSL validity checking

A formula φ is *valid*, written $\models \varphi$ when it is satisfied by all models. It is *satisfiable* when there is at least one model that satisfies it. It is *invalid* or *unsatisfiable* if it is satisfied by no models. This implies that φ is satisfiable if and only if $\neg\varphi$ is unsatisfiable. Similarly, φ is unsatisfiable if and only if $\neg\varphi$ is valid. And φ is valid if and only if $\neg\varphi$ is unsatisfiable.

There are different approaches to checking PSL formulas. The main approaches are:

1. Translate the PSL formula $\varphi^{(\mathsf{psl})}$ into an LTL formula $\varphi^{(\mathsf{ltl})}$, and then:
   (a) either use an LTL model-checker to test for the validity of $\varphi^{(\mathsf{ltl})}$,
   (b) or use the Tableau method on $\varphi^{(\mathsf{ltl})}$.
2. Translate the negated PSL formula $\neg\varphi^{(\mathsf{psl})}$ directly into a Büchi automaton and emptyness checking.

The first approach has two phases: translation into LTL, and LTL validity checking. Alternative (a) is explained in Subsection 9.3. Alternative (b), first proposed in [Wol85] consists of building a *tableau* from the negation of the formula, this is, a graph whose nodes are sets of sub-formulas, according to certain construction rules and then iteratively eliminating nodes that generate contradictions or violate temporal properties. If the entire tableau is eliminated, the original formula is valid.

In the second approach, the PSL formula $\neg\varphi^{(\mathsf{psl})}$ is translated into a Büchi automaton and then a it is checked whether there are any legal behaviours that go through some accepting states of the automaton. If this is the case, then the language of $\neg\varphi^{(\mathsf{psl})}$ is not empty, and therefore $\varphi^{(\mathsf{psl})}$ is not valid. On the other hand, if no legal behaviour goes through an accepting state, then the language of $\neg\varphi^{(\mathsf{psl})}$ is empty, which makes $\varphi^{(\mathsf{psl})}$ valid.

The first approach has the advantage that there are several off-the-shelf tools that perform LTL model-checking and which have been optimized. The disadvantage is that PSL is strictly more expressive than LTL: as shown in [Wol83], LTL cannot express certain ω-regular properties such as "p occurs at every even point in time, and may or may not occur at odd points in time." These properties, however, are expressible in PSL. Hence, it is not possible to translate all PSL formulas into LTL formulas. The only solution, if this approach is used, is to restrict the specifications to a subset of PSL, and translate commonly used PSL expression patterns. This is the approach followed by [CRT08] and is used by the NuSMV tool described below.

The second approach has the advantage that Büchi automata do capture all ω-regular properties and thus, all of PSL can be handled. The disadvantage is that there are very few tools that support such translation and emptyness check. The Spot tool described below follows this approach.

9.2 PSL verifiers and translators

Table 3 on page 51 shows a summary of tools available which support verification and/or translation of PSL formulas. We now describe them.

Tool name	**Verifier or translator**	**PSL support**					**Output**	**Licence**
		Standard syntax	**SEREs**	**LTL**	**OBE (CTL)**	**Clocked expr.**		
NuSMV	Verifier and translator	Yes	Partial (†)	All	Yes	Yes	XML traces for counter-examples. Verifier output has no machine readable format.	LGPL v2.1
Spot	Translator	No	All (*)	All	No	No	Spin never claim	GPL v3
psl2ba	Translator	No	All (*)	All	No	No	SMV model	BSD 3-clause
IBM RuleBase SixthSense Edition	?	?		?	?	?	?	Proprietary, Commercial

Table 3: PSL tools. Legend: (†) Supports the subset of SEREs for which a translation to LTL is possible. (*) Supports all SERE operators, but may have non-standard semantics.

- NuSMV (http://nusmv.fbk.eu/) is a successor to the SMV model-checker which introduced BDD-based model-checking. It includes partial support of PSL, but not the full support, because, as explained in Subsection 9.1, its approach to PSL model checking is to translate PSL specifications into LTL, but because LTL is less expressive than PSL, some PSL formulas are not handled, in particular some SERE expressions are ignored. Aside from ω-regular properties, NuSMV does not support either the OBE extension of PSL, nor clocked expressions. However, many common PSL patterns are handled, as described in [CRT08]. A significant advantage of NuSMV is that it supports symbolic model checking with either BDDs or SAT solvers. The user simply passes a parameter to the model-checker saying whether to use BDDs or one of the two SAT solvers included. The output produced by NuSMV is not ina machine readable format, but counterexamples can be generated in XML format. NuSMV's licence is the LGPL.

- Spot (http://spot.lip6.fr/wiki/SpotWiki) is model checking library written in C++ with support for PSL. Since it translates PSL formulas directly into Büchi automata, it is able to handle all SEREs. Nevertheless, like NuSMV, it does not support the OBE extension nor clocked expressions. It is in active development, and full compliance with PSL standard semantics is unclear. It's licence is the GPL.

- PSL2BA (https://code.google.com/p/psl2ba/) is an experimental translator from PSL to Büchi automata. It does not have significant documentation, so full support for PSL remains unclear. It's output representation used as input to NuSMV. It's licence is the New BSD License.

- IBM RuleBase SixthSense Edition (https://www.research.ibm.com/haifa/projects/verification/Formal_Methods-Home/, http://researcher.watson.ibm.com/researcher/view_project.php?id=2987) is a commercial, proprietary tool that supports PSL. No public documentation is available at the time of this writing, so our knowledge of its capabilities is limited.

9.3 Using a PSL model-checker to check for validity

The general approach to model-checking takes some model M and a temporal formula φ and tries to establish whether M satisfies φ ($M \vDash \varphi$). To do this, the model-checker must establish that the language of M (the set of possible behaviours of M, written $\mathcal{L}(M)$) is a subset of the language of the formula $\mathcal{L}(\varphi)$. To establish $\mathcal{L}(M) \subseteq \mathcal{L}(\varphi)$, it is enough to check $\mathcal{L}(M) \cap \overline{\mathcal{L}(\varphi)} = \varnothing$, this is, that there are no behaviours of M which are not behaviours of φ. In a typical temporal logic we have that $\overline{\mathcal{L}(\varphi)} = \mathcal{L}(\neg\varphi)$, so we need to check that $\mathcal{L}(M) \cap \mathcal{L}(\neg\varphi) = \varnothing$. Usually the model-checker transforms the formula $\neg\varphi$ into a (Büchi) automaton $A_{\neg\varphi}$, then computes some product between automata $M \otimes A_{\neg\varphi}$. If this automaton has legal behaviours, then $\mathcal{L}(M) \cap \mathcal{L}(\neg\varphi) \neq \varnothing$, and therefore $\mathcal{L}(M) \nsubseteq \mathcal{L}(\varphi)$; otherwise $\mathcal{L}(M) \cap \mathcal{L}(\neg\varphi) = \varnothing$ and $\mathcal{L}(M) \subseteq \mathcal{L}(\varphi)$.

When we are trying to check for the validity of a formula or its satisfability, we only have the formula, but the model-checker expects a model M as input as well. Nevertheless, a model-checker can be used for satisfability checking (and therefore validity checking as well). This can be accomplished, as described in [RV10] by providing a "universal model" W, which contains all possible behaviours. Typically such a model would consist of a single accepting states with all transitions labelled with each atomic proposition appearing in the formula to be checked.

For example, suppose we want to check whether the LTL formula $\varphi_0 \stackrel{def}{=} \mathsf{G}\,(a \vee b) \rightarrow \mathsf{F}\,b$ is valid. In this formula, the set of atomic propositions is $\{a, b\}$. Hence the universal model (Büchi automaton) W_0 would consist of a single accepting state s_0 and transitions $s_0 \xrightarrow{a} s_0$ and $s_0 \xrightarrow{b} s_0$. The model checker would translate $\neg\varphi_0 = \mathsf{G}\,(a \vee b) \wedge \neg\,\mathsf{F}\,b$ into a Büchi automaton $A_{\neg\varphi_0}$ and compute the synchronous product $W_0 \otimes A_{\neg\varphi_0}$. But it turns out that the sequence $aaa... \in \mathcal{L}(\neg\varphi_0)$ so $\mathcal{L}(W_0) \cap \mathcal{L}(\neg\varphi_0) \neq \varnothing$ and so $\neg\varphi_0$ is satisfiable, which makes φ_0 invalid. Using NuSMV, the universal model can be expressed simply by declaring a boolean variable for each atomic proposition as shown in Figure 15 on page 53. As expected, feeding this model to NuSMV reports that the formula is false.

Going back to our original example, recall that in Subsection 6.3 we had transformed the PSL guarantees for the Academic, Conference and Repository subcapsules of the Academia composite capsule as follows:

```
MODULE main
  VAR
    a : boolean;
    b : boolean;
  LTLSPEC (G (a|b) -> F b)
```

Figure 15: NuSMV model to check the LTL validity of φ_0.

```
MODULE main
  VAR
    in_conn3_funds : boolean;
    out_conn2_paper : boolean;
    in_conn4_paper : boolean;
    in_conn5_paper : boolean;
  PSLSPEC ( ( G (in_conn3_funds -> F out_conn2_paper)
            & G (out_conn2_paper -> F in_conn4_paper)
            & G (in_conn4_paper -> F in_conn5_paper) )
            -> G (in_conn_3_funds -> F in_conn_5_paper) )
```

Figure 16: NuSMV model for checking the validity of the Academia1 guarantee.

$$
\begin{aligned}
\mathsf{ghat}[\![G_1]\!]_{\mathsf{Academia1,Academic1}} &\stackrel{def}{=} \mathsf{G}(\mathsf{in} : \mathsf{conn3.funds} \rightarrow \mathsf{F}\,\mathsf{out} : \mathsf{conn2.paper}) \\
\mathsf{ghat}[\![G_2]\!]_{\mathsf{Academia1,Conference1}} &\stackrel{def}{=} \mathsf{G}(\mathsf{out} : \mathsf{conn2.paper} \rightarrow \mathsf{F}\,\mathsf{in} : \mathsf{conn4.paper}) \\
\mathsf{ghat}[\![G_3]\!]_{\mathsf{Academia1,Repository1}} &\stackrel{def}{=} \mathsf{G}(\mathsf{in} : \mathsf{conn4.paper} \rightarrow \mathsf{F}\,\mathsf{in} : \mathsf{conn5.paper}) \\
\mathsf{ghat}[\![G]\!]_{\mathsf{Academia1,Academia1}} &\stackrel{def}{=} \mathsf{G}(\mathsf{in} : \mathsf{conn3.funds} \rightarrow \mathsf{F}\,\mathsf{in} : \mathsf{conn5.paper})
\end{aligned}
$$

To perform the compositional inference, we need to establish that the formula $G_1' \wedge G_2' \wedge G_3' \rightarrow G$ where $G_1' \stackrel{def}{=} \mathsf{ghat}[\![G_1]\!]_{\mathsf{Academia1,Academic1}}$, $G_2' \stackrel{def}{=} \mathsf{ghat}[\![G_2]\!]_{\mathsf{Academia1,Conference1}}$, $G_3' \stackrel{def}{=} \mathsf{ghat}[\![G_3]\!]_{\mathsf{Academia1,Repository1}}$ and $G \stackrel{def}{=} \mathsf{ghat}[\![G]\!]_{\mathsf{Academia1,Academia1}}$. We can check the validity of this formula by feeding NuSMV with the model shown in Figure 16 on page 53.

10 Summary and future work

We have explored a theoretical framework to support compositional analysis based on assume/guarantee contract-based reasoning developed mainly in [BDH+12a, BDH+12b]. We have shown how a small adaptation of PSL conforms to the notion of specification theory required by the theoretical framework and how its results can be applied to PSL specifications. This in turn has led us to propose some compositional verification algorithms based on this framework and which can be integrated into the RTEdge™ platform using third-party, off-the-shelf PSL verifiers.

The theoretical framework from [BDH+12a, BDH+12b] gave us the foundation for this work, as it covered the essential aspects. Nevertheless, that framework assumes one underlying specification language. In our case, we have had to deal with two languages, the modelling language RTEdge™, and the specification language, PSL. Therefore we had to take this into consideration and put in place the necessary machinery to support a modelling language separate from a specification language. Furthermore, we have had to adapt the specification language to properly express properties of the behaviour of RTEdge™ models.

In addition to these changes, we have extended the theory by defining commutative monoid and standard specification theories, which give the contract theory a logical structure and allow us to generalize the results from [BDH+12a, BDH+12b] in order to support n-ary contract composition and quotiening of contracts.

While the theory provides us with a framework to support compositional and incremental analysis, there are several issues that we have not addressed and which are left for future work. The main open problems include:

- Regarding extensions to the specification language:
  - How to better capture asynchronous interaction in PSL specifications.
  - How to support properties about the contents of port's message queues.
- Regarding the implementation of the framework:
  - How to translate (atomic) RTEdge™ capsules into PSL specifications so that we can use the proposed algorithms. This translation must be such that it satisfies the requirements of Definition 24 as the correctness of the algorithms rely on these.
  - Optimize the algorithms, particularly those that perform incremental analysis. A syntactic analysis on the model structure can simplify the structure of the contract composition, reducing the time of validity checking done.
  - Identify potential limitations, if any, of these algorithms with respect to different kinds of properties, *i.e.*, safety, liveness and fairness.

Acknowledgements

This research was funded by the Natural Sciences and Engineering Research Council of Canada (NSERC) under the NSERC Engage Grant partnership program (EGP-445280-12). Edgewater Computer Systems, Inc. provided in-kind contributons in the form of a licence of RTEdge™ as well as technical support, and research support by staff. We would like to thank Serban Gheorghe, Vice-President of Technology at Edgewater, with whom we held bi-weekly meetings discussing the project for the length of its duration. Serban was fundamental in helping us understand the RTEdge™ language and platform, and provided valuable feedback and encouragement. Andrzej Wąsowski, one of the coauthors of [BDH+12a], provided useful comments and help with an assesment of the theoretical framework's applicability to PSL and RTEdge™. We also thank David Andrews whose questions pointed out some potential problems with our original solution to the protocol-interface conformance problem. Finally, Juergen Dingel also provided many helpful questions and comments, as well as support and encouragement for this work.

References

[Acc04] Accellera. *Property Specification Language*. Accellera, http://www.eda.org/ieee-1850/, version 1.1 edition, 9 June 2004.

[BCC97] S. Berezin, S. Campos, and E. M. Clarke. Compositional reasoning in model checking. In *COMPOS: Int. Symp. on Compositionality: The Significant Difference*. LNCS, 1997.

[BDH+12a] S. S. Bauer, A. David, R. Hennicker, K. G. Larsen, A. Legay, U. Nyman, and A. Wasowski. Moving from specifications to contracts in component-based design. In Juan de Lara and Andrea Zisman, editors, *Fundamental Approaches to Software Engineering - 15th International Conference, FASE 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012. Proceedings*, volume 7212 of *Lecture Notes in Computer Science*, pages 43–58. Springer, 2012.

[BDH+12b] S. S. Bauer, A. David, R. Hennicker, K. G. Larsen, A. Legay, U. Nyman, and A. Wasowski. Moving from specifications to contracts in component-based design. Tech. report 1201, Institut für Informatik, Lidwig-Maximilians-Universität München, January 2012.

[BK08] C. Baier and J-P. Katoen. *Principles of Model Checking*. MIT Press, 2008.

[BO01] V. A. Braberman and A. Olivero. Extending timed automata for compositional modeling healthy timed systems. *MTCS 2001 Models for Time-Critical Systems*, 52(3):227–245, August 2001.

[CCJK12] T. Chen, C. Chilton, B. Jonsson, and M. Z. Kwiatkowska. A compositional specification theory for component behaviours. In Helmut Seidl, editor, *Programming Languages and Systems - 21st European Symposium on Programming, ESOP 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012. Proceedings*, volume 7211 of *Lecture Notes in Computer Science*, pages 148–168. Springer, 2012.

[CE81] E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In Dexter Kozen, editor, *Proc. of the Workshop on Logics of Programs*, volume 131 of *LNCS*, pages 52–71. Springer, 1981.

[CGP99] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, Cambridge, Massachusetts, 1999.

[Cha94] E. Chang. *Compositional verification of reactive and real-time systems*. PhD thesis, Stanford University, 1994.

[CLM89] E. M. Clarke, D. E. Long, and K. L. McMillan. Compositional model checking. In *Proceedings, 4th Annual Symp. on Logic in Computer Science*, pages 353–362, Asilomar Conference Center, Pacific Grove, California, 5–8 June 1989. IEEE Computer Society Press.

[CRT08] A. Cimatti, M. Roveri, and S. Tonetta. Symbolic Compilation of PSL. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 27(10):1737–1750, 2008.

[dAH01] L. de Alfaro and T. A. Henzinger. Interface Automata. *SIGSOFT Software Engeering Notes*, 26(5):109–120, September 2001.

[DG99] M. Dam and D. Gurov. Compositional verification of CCS processes. In Dines Bjørner, Manfred Broy, and Alexandre V. Zamulin, editors, *Perspectives of System Informatics*, volume 1755 of *LNCS*, pages 247–256. Springer, 1999.

[DLP04] A. Duret-Lutz and D. Poitrenaud. SPOT: An extensible model checking library using transition-based generalized Büchi automata. In Doug DeGroot, Peter G. Harrison, Harry A. G. Wijshoff, and Zary Segall, editors, *Proc. of the Int. Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS 2004)*, pages 76–83. IEEE Computer Society, 2004.

[ELSS07] A. Easwaran, I. Lee, I. Shin, and O. Sokolsky. Compositional schedulability analysis of hierarchical real-time systems. In *Proc. of the Tenth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'07)*, pages 274–281, Santorini Island, Greece, May 2007. IEEE Computer Society.

[Ghe11] S. Gheorghe. Integration of Formal Model Checking with the RTEdge™ AADL Microkernel. Tech. Report 2011-01-2531, SAE International/Edgewater Computer Systems Inc., 18 October 2011.

[Gie00] H. Giese. Contract-based component system design. In *Proc. of the 33rd Annual Hawaii International Conference on System Sciences (HICSS-33)*, 2000.

[GL94] O. Grumberg and D. E. Long. Model checking and modular verification. *ACM Transactions on Programming Languages and Systems*, 16(3):843–871, May 1994.

[Hol04] G. J. Holzmann. *The SPIN Model Checker - Primer and Reference manual*. Addison-Wesley, 2004.

[IBM05] IBM. *General Description Language*. IBM, 9 March 2005.

[IEE01] IEEE Computer Society. IEEE Standard Verilog® Hardware Description Language, IEEE Standard 1364™-2001, 28 September 2001.

[IEE05] IEEE Computer Society. IEEE Standard for Property Specification Language (PSL). IEEE Standard 1850™-2005, 17 October 2005.

[IEE09] IEEE Computer Society. IEEE Standard VHDL Language Reference Manual, IEEE Standard 1076™-2008, 26 January 2009.

[IEE12a] IEEE Computer Society. IEEE Standard for Property Specification Language (PSL). IEEE Standard 1850™-2010, June 2012.

[IEE12b] IEEE Computer Society. IEEE Standard for the SystemC Language, IEEE Standard 1666™-2011, January 2012.

[IEE13] IEEE Computer Society. IEEE Standard for SystemVerilog – Unified Hardware Design, Specification, and Verification Language, IEEE Standard 1800™-2012, 21 February 2013.

[Jon90] B. Jonsson. A hierarchy of compositional models of I/O-automata (extended abstract). In Branislav Rovan, editor, *Mathematical Foundations of Computer Science 1990*, volume 452 of *LNCS*, pages 347–354, Banská Bystrica, Czechoslovakia, 27–31 August 1990. Springer.

[KL04] D. K. Kaynar and N. A. Lynch. Decomposing verification of timed I/O automata. In Yassine Lakhnech and Sergio Yovine, editors, *Proc. of Formal techniques, modelling and analysis of timed and fault-tolerant systems, joint international conferences on formal modelling and analysis of timed systems, FORMATS 2004 and formal techniques in real-time and fault-tolerant systems, FTRTFT 2004*, volume 3253 of *LNCS*, pages 84–101. Springer, 2004.

[KV97] O. Kupferman and M. Y. Vardi. Modular model checking. In Willem P. de Roever, Hans Langmaack, and Amir Pnueli, editors, *COMPOS: International Symposium on Compositionality: The Significant Difference*, volume 1536 of *LNCS*, pages 381–401. Springer, 1997.

[LA10] M. D. Lee and P. R. Argenio. A refinement based notion of non-interference for interface automata: Compositionality, decidability and synthesis. In *Proc. of the XXIX International Conference of the Chilean Computer Science Society (SCCC 2010)*, pages 280–289. IEEE Computer Society, IEEE Computer Society, 2010.

[LG98] K. Laster and O. Grumberg. Modular model checking of software. In Bernhard Steffen, editor, *Proc. of Tools and algorithms for construction and analysis of systems, (TACAS '98)*, volume 1384 of *LNCS*, pages 20–35. Springer, 1998.

[LS89] N. Lynch and E. W. Stark. A proof of the Kahn principle for Input/Output automata. *Information and Computation*, 82:81–92, 1989.

[LSV03] N. A. Lynch, R. Segala, and F. W. Vaandrager. Hybrid I/O automata. *Information and Computation*, 185(1):105–157, 2003.

[LT87] N. A. Lynch and M. R. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *Proc. of the 6th annual ACM Symposium on Principles of Distributed Computing (PODC '87)*, PODC '87, pages 137–151, New York, NY, USA, 1987. ACM.

[Pnu77] A. Pnueli. The temporal logic of programs. In *Proc. of the 18th IEEE Symp. on the Foundations of Comp. Sci. (FOCS '77)*, pages 46–57, Providence, Rhode Island, October 31–November 2 1977. IEEE, IEEE Computer Society Press.

[Pnu84] A. Pnueli. In transition from global to modular temporal reasoning about programs. In K. Apt, editor, *Logic and Models of Concurrent Systems*, volume 13 of *NATO ASI series in Computer and System Sciences*, pages 123–144, New York, 1984. Springer-Verlag.

[Pos13a] E. Posse. A formal semantics for RTEdge™. Tech. report, Edgewater Computer Systems Inc., Edgewater Computer Systems Inc. 1125 Innovation Drive, Ottawa, Ontario, Canada - K2K 3G6., July 2013. `http://www.edgewater.ca`.

[Pos13b] E. Posse. A formal semantics for RTEdge™. Tech. report 2013-606, School of Computing – Queen's University, August 2013. `http://research.cs.queensu.ca/TechReports/Reports/2013-606.pdf`.

[PV02] F. Plasil and S. Visnovsky. Behavior Protocols for Software Components. *IEEE Transactions on Software Engineering*, 28(11):1056–1076, November 2002.

[RR01] S. K. Rajamani and J. Rehof. A behavioral module system for the π-calculus. In *International Symposium on Static Analysis (SAS) , Paris, France*, volume 2126 of *Lecture Notes in Computer Science*, pages 375–394. Springer-Verlag, July 2001.

[RV10] K. Y. Rozier and M. Y. Vardi. LTL Satisfiability Checking. *International Journal on Software Tools for Technology Transfer (STTT)*, 12(2):123–137, May 2010.

[SAE12] SAE International. Architecture Analysis & Design Language (AADL). SAE Standard AS5506b, 10 September 2012.

[SC85] A. P. Sistla and E. M. Clarke. The Complexity of Propositional Linear Temporal Logics. *Journal of the ACM*, 32(3):733–749, 1985.

[Shi06] I. Shin. *Compositional framework for real-time embedded systems.* PhD thesis, University of Pennsylvania, January 01 2006.

[Sip97] M. Sipser. *Introduction to the Theory of Computation.* PWS Publishing, 1997.

[SS98] E. W. Stark and S. A. Smolka. Compositional analysis of expected delays in networks of probabilistic I/O automata. In *Proc. of the 13th Annual IEEE Symposium on Logic in Computer Science (LICS'98)*, pages 466–477, Indianapolis, Indiana, USA, June 1998. IEEE Computer Society.

[Sta00] E. W. Stark. Compositional performance analysis using probabilistic I/O automata. In Catuscia Palamidessi, editor, *Proc. of CONCUR 2000 - concurrency theory, 11th international conference*, volume 1877 of *LNCS*, pages 25–28. Springer, 2000.

[Wol83] P. Wolper. Temporal Logic Can Be More Expressive. *Information and Control*, 56:72–99, 1983.

[Wol85] P. Wolper. The Tableau Method for Temporal Logic: An Overview. *Logique et Analyse*, 28(110–111):119–136, June–September 1985.

[WSS94] S-H. Wu, S. A. Smolka, and E. W. Stark. Composition and behaviors of probabilistic I/O automata. In Bengt Jonsson and Joachim Parrow, editors, *Proc. of CONCUR '94: concurrency theory*, volume 836 of *LNCS*, pages 513–528, Uppsala, Sweden, 22–25August 1994. Springer-Verlag.

[WW09] H. Wehrheim and D. Wonisch. Compositional CSP traces refinement checking. *Electr. Notes Theor. Comput. Sci.*, 250(2):135–151, 2009.

Index

A RTEdge^{TM} description and abstract syntax

In this appendix we present a subset of the abstract syntax of RTEdge^{TM} which is relevant to this work. For a more thorough description of the syntax and the formal semantics of RTEdge^{TM} we refer the reader to [Pos13b, Pos13a].

A.1 Informal description

RTEdge^{TM} is a language that can be used to describe concurrent, reactive, real-time systems. In RTEdge^{TM}, a system is a collection of interconnected *components* or *processes* called *capsules*. Each capsule is an *active object* with attributes and reactive behaviour. A capsule executes concurrently with the other components in the system. Capsules interact with other components by sending messages or *signals* over connections (also called *connectors*). Each capsule has a well-defined *interface* which consists of a set of ports through which signals are sent and received. Connectors link ports between different capsules. The reactive behaviour of capsules is defined by a certain kind of state machines. Communication is *asynchronous*: the sending of a message is non-blocking, so the sender doesn't wait for the message to be delivered. Capsules can be composed and grouped together to define a hierarchical structure.

The core elements of the RTEdge^{TM} language are:

- Protocols
- Interfaces
- Atomic Capsules with State Machines
- Composite Capsules
- Proxy Capsules
- External Task Capsules
- Timers
- Applications

Informally a *protocol* defines a set of input and output signals which may be transmitted between capsules.

An *interface* defines a collection of *named ports*, each of which has a protocol and can be either a *base port* or a *conjugate port*. In a base port, input and output signals of the corresponding protocol, are treated by the owning capsule as inputs and outputs respectively, whereas in a conjugate port the roles are flipped: input signals of the protocol are treated by the capsule as outputs and output signals in the protocol are treated as inputs.

An *atomic capsule* defines a process or active object with data attributes and a behaviour and has a specific interface. The behaviour is defined by a *state machine*. RTEdge^{TM} state machines are flat (no hierarchical states) and divide the states into two groups: stable states and transient states. Stable states are states where the capsule is at rest waiting for external input signals on its ports. Hence transitions emanating from stable states are annotated with *input or external event triggers*. Transient states are intermediate states which may have actions or activities associated to them. These activities are written in an underlying *action language*, which in the case of the RTEdge^{TM} platform is C++. Actions are parametrized with the data attributes of the capsule object and with the last message received. Transitions emanating from a transient state can be labelled with a value or *action completion code*, which is used as an *activity completion trigger*, allowing internal choice. Transitions can be annotated with *output statements*, which send output signals through the capsule ports to other capsules. These state machines have a *run-to-completion semantics*: when the capsule is on a stable state, the arrival of an input on a port results in a chain of transitions being followed according to the transition triggers, possibly going through transient states and ending in a stable state. If an input signal arrives and the capsule is not on a stable state, the signal will be queued in its port until the capsule can handle it. Thus, each port has its own FIFO queue. A port might be marked as *deferred* in a stable state. If an input arrives on that port when the system is in the stable state that defers it, the message will not be processed, and it will simply remain in its queue until it can be consumed in another stable state. If

input arrives at a port which is not deferred and the current stable state doesn't have a transition with a trigger for that port and signal, there are two possible behaviours: the signal can be ignored, or an error can be issued. If more than one transition is enabled in a stable state, the tie is broken by the relative priorities of the signals. These priorities are assigned statically by RTEdge$^{\mathrm{TM}}$ during schedulability analysis, to ensure that required deadlines are met.

A *composite capsule* defines a group of interconnected capsules (atomic or composite) and has an interface. It serves a the basic structuring construct in the language providing a hiding and encapsulation construct so that the only way to access the composite capsule's sub-components is through its interface. Informally, the capsules within a composite capsule execute concurrently, although the platform implementation may schedule the transitions within the same thread. Since ports are queues, the basic communication mechanism is asynchronous message passing.

A *proxy capsule* is a special kind of atomic capsule which has "OS ports", this is, ports that allow the capsule to interact with software outside of the application.

An *external task capsule* is also a special kind of capsule and it doesn't represent a component within the application, but rather it is used to represent external components with which the application may interact. External capsules can only be connected to proxy capsules.

A *timer* is a special kind of primitive component that issues a given signal periodically. This signal is received by every capsule which has subscribed to the timer's *service*.

An *application* is the top level component of the language. It can be seen as a special composite capsule which groups together all components and elements.

A.2 Formal description

In this appendix we omit the detailed definitions of types, state machines, proxies, external tasks and applications, as they are not necessary to describe the framework. We refer the reader to [Pos13a] for a complete definition of these constructs.

We assume a set **Names** of all possible names, a set **Types** of all possible data-types, a set **Values** of all possible data values over the given data-types.

Definition 30 (**Protocols**)**.** A *protocol* is a tuple $(I, O, type)$ where:

- $I \subseteq$ **Names** is a set of *input signals*
- $O \subseteq$ **Names** is a set of *output signals*
- $I \cap O = \varnothing$
- $type : I \cup O \to$ **Types** is a typing function, assigning a type to each signal

We call **Protocols** the set of all possible protocols. Given a protocol $R = (I, O, type)$ we define

- $\mathsf{isignals}(R) \stackrel{def}{=} I$
- $\mathsf{osignals}(R) \stackrel{def}{=} O$
- $\mathsf{typmap}(R) \stackrel{def}{=} type$

Definition 31 (**Interfaces**)**.** An *interface* F is a tuple $(P, L, prot, kind)$ where

- $P \subseteq$ **Names** is a set of *port names*,
- $L \subseteq$ **Protocols** is a set of protocols,
- $prot : P \to L$ is an assignment of protocols to ports,
- $kind : P \to \{\mathsf{base}, \mathsf{conj}\}$ is an assignment of kinds to ports

We call **Interfaces** the set of all possible interfaces. We define

- $\mathsf{ports}(F) \stackrel{def}{=} P$
- $\mathsf{protocols}(F) \stackrel{def}{=} L$
- $\mathsf{protmap}(F) \stackrel{def}{=} prot$
- $\mathsf{kindmap}(F) \stackrel{def}{=} kind$

Definition 32 (**Atomic capsules**)**.** An *atomic capsule* K is a tuple (F, Σ, A, M) where

- $F = (P, L, prot, kind) \in$ **Interfaces** is an *interface* (see Definition 31),
- $\Sigma = (V, type : V \to$ **Types**$)$ is variable signature where the variables are called *attributes*, (see [Pos13a], Section 2.1, Definition 8)
- $A \subseteq$ **Activities**$_\Sigma$ is a of of *actions* or *activities*, (see [Pos13a], Section 2.5, Definition 16)
- $M = (S, i, Q, F, def, \Sigma, A, act, T) \in$ **StateMachines** is a state machine (see [Pos13a], Section 2.5, Definition 19)

We call **Atomic** the set of all possible atomic capsules. We define:

- $\mathsf{ports}(K) \stackrel{def}{=} \mathsf{ports}(F) = P$
- $\mathsf{interface}(K) \stackrel{def}{=} F$
- $\mathsf{attributes}(K) \stackrel{def}{=} \Sigma$
- $\mathsf{activities}(K) \stackrel{def}{=} A$
- $\mathsf{statemachine}(K) \stackrel{def}{=} M$

Definition 33 (**Composite capsules**)**.** A *composite capsule* K is a tuple $(F, H, R, role, C, link)$ where

- $F = (P, L, prot, kind) \in$ **Interfaces** is an *interface* (see Definition 31),
- $H \subseteq$ **Capsules** is a finite set of (atomic or composite) capsules called *parts*, subject to the condition that $K \notin H$ and K is not a sub-part (transitively) of any of its parts,
- $R \subseteq$ **Names** is a finite set of *role names*, such that $\mathsf{self} \notin R$,
- $role : R \uplus \{\mathsf{self}\} \to H \uplus \{K\}$ is a map associating each role with a capsule, where $role(\mathsf{self}) \stackrel{def}{=} K$, and for all $r \in R$, $role(r) \neq K$,
- $C \subseteq$ **Names** is a finite set of *connector names*,
- $link : C \to \mathsf{connpts}(K) \times \mathsf{connpts}(K)$ is map assigning each connector name to a *link* $(b_1, b_2) \in \mathsf{connpts}(K) \times \mathsf{connpts}(K)$ where $\mathsf{connpts}(K)$ denotes the set of all *connection points* of K and is defined as

$$\mathsf{connpts}(K) \stackrel{def}{=} \{(\mathsf{self}, p) \mid p \in P\} \cup \bigcup_{r \in R} \{(r, p) \mid p \in \mathsf{ports}(role(r))\}$$

  We write $r.p$ for a connection point (r, p). For any connector $c \in C$, $link(c)$ must satisfy the following conditions:

  - for any connection point $r.p$, $(r.p, r.p) \notin link(c)$ (*i.e.*, $link(c)$ must be *irreflexive*, a port cannot be connected to itself),
  - for any connection point $r_1.p_1$, there is at most one connection point $r_2.p_2$ such that $(r_1.p_1, r_2.p_2) \in link(c)$ (i.e., $\mathsf{links}(c)$ must be a *partial* or *total function*)

- for any connection point $r_2.p_2$, there is at most one connection point $r_1.p_1$ such that $(r_1.p_1, r_2.p_2) \in link(c)$ (i.e., $link(c)$ must be a *one-to-one mapping*)
- whenever $link(c) = (r_1.p_1, r_2.p_2)$ such that $r_1 \neq \mathsf{self}$ and $r_2 \neq \mathsf{self}$, $prot_1(p_1) = prot_2(p_2)$ and either
  - $kind_1(p_1) = \mathsf{base}$ and $kind_2(p_2) = \mathsf{conj}$ or
  - $kind_1(p_1) = \mathsf{conj}$ and $kind_2(p_2) = \mathsf{base}$

  where $prot_i = \mathsf{protmap}(\mathsf{interface}(role(r_i)))$ and $kind_i = \mathsf{kindmap}(\mathsf{interface}(role(r_i)))$ for $i \in \{1, 2\}$ (*i.e.*, a connection between internal parts can only be between a base port and a conjugated port)
- whenever $link(c) = (r_1.p_1, r_2.p_2)$ such that $r_1 = \mathsf{self}$ and $r_2 \neq \mathsf{self}$ or $r_1 \neq \mathsf{self}$ and $r_2 = \mathsf{self}$, $prot_1(p_1) = prot_2(p_2)$ and either
  - $kind_1(p_1) = \mathsf{base}$ and $kind_2(p_2) = \mathsf{base}$ or
  - $kind_1(p_1) = \mathsf{conj}$ and $kind_2(p_2) = \mathsf{conj}$

  where $prot_i = \mathsf{protmap}(\mathsf{interface}(role(r_i)))$ and $kind_i = \mathsf{kindmap}(\mathsf{interface}(role(r_i)))$ for $i \in \{1, 2\}$ (*i.e.*, a connection between a port of the composite capsule and a port of a sub-capsule must be of the same kind)

We call **Composite** the set of all possible composite capsules and **Capsules** $\stackrel{def}{=}$ **Atomic** $\cup$ **Composite** the set of all capsules. We define

- $\mathsf{ports}(K) \stackrel{def}{=} \mathsf{ports}(F) = P$
- $\mathsf{interface}(K) \stackrel{def}{=} F$
- $\mathsf{parts}(K) \stackrel{def}{=} H$
- $\mathsf{roles}(K) \stackrel{def}{=} R$
- $\mathsf{connectors}(K) \stackrel{def}{=} C$
- $\mathsf{links}(K) \stackrel{def}{=} (\bigcup_{c \in C} link(c)) \cup (\bigcup_{c \in C} \{(r_2.p_2, r_1.p_1) \mid (r_1.p_1, r_2.p_2) \in link(c)\})$
- $\mathsf{linksmap}(K) \stackrel{def}{=} link$

B PSL core syntax and semantics

PSL consists of four language *layers*:

- The *boolean layer*, to describe state or event conditions at a given point in the execution of a system.
- The *temporal layer*, to describe properties and behaviour of the system over time.
- The *verification layer*, to describe verification units and provide directives and intended for a verification tool.
- The *modelling layer*, to describe the model under consideration.

The temporal layer consists of the Foundation Language (FL) and an Optional Branching Extension (OBE). The Foundation Language consists of:

- Sequential Extended Regular Expressions (SEREs)
- Linear Temporal Logic (LTL) [Pnu77]
- Operators to combine or embed SEREs into LTL formulas
- Derived operators

The Optional Branching Extension consists mostly of Computation Tree Logic (CTL) [CE81].

The verification layer defines:

- Verification directives: constructs that can be used to declare specifications and define, for example, assertions, assumptions, fairness constraints, etc.
- Verification units: groups of declarations and directives with a given name and possibly a binding to a model artifact.

The modelling layer is intended to provide the means to describe system behaviour, but it is specific to the *flavour*, which can be one of the well known hardware description languages: Verilog [IEE01], SystemVerilog [IEE13], VHDL [IEE09], SystemC [IEE12b] or GDL [IBM05]. In this report we consider the modelling language to be $\mathsf{RTEdge}^{\mathrm{TM}}$.

In this appendix we describe only the syntax of the core, *unclocked* Foundation Language (FL) fragment of PSL [IEE12a, IEE05, Acc04]. For a description of the syntax and semantics (formal and informal) of the full PSL we refer the reader to [IEE12a, IEE05, Acc04].

B.1 Syntax

Notation 4. PSL is defined over a given set **AP** of *atomic propositions*. We write $\mathbf{BoolExpr}_{\mathbf{AP}}$ for the set of boolean expressions over **AP**, or simply **BoolExpr** if **AP** is clear from the context. We use $b, b_1, b_2, ...$ as meta-variables that range over the set **BoolExpr**. We define below the set **SERE** of SEREs and the set **FL** of FL formulas. We use $r, r_1, r_2, ...$ as meta-variables ranging over **SERE**, and $\varphi, \varphi_1, \varphi_2, ...$ as meta-variables ranging over **FL**.

Definition 34 (**Boolean expressions** [IEE12a, IEE05, Acc04])**.** The set $\mathbf{BoolExpr}_{\mathbf{AP}}$ of boolean expressions over the set **AP** of atomic propositions is defined as the smallest set that satisfies the following:

- $\mathsf{true} \in \mathbf{BoolExpr}$
- $\mathsf{false} \in \mathbf{BoolExpr}$
- If $p \in \mathbf{AP}$ is an atomic proposition, then $p \in \mathbf{BoolExpr}$
- If $b, b_1, b_2 \in \mathbf{BoolExpr}$ then
  - $\neg b \in \mathbf{BoolExpr}$

– $b_1 \wedge b_2 \in$ **BoolExpr**

Definition 35 (**Sequential Extended Regular Expressions - SEREs** [IEE12a, IEE05, Acc04])**.** The set **SERE** is defined as the smallest set that satisfies the following:

- If $b \in$ **BoolExpr** then $b \in$ **SERE**.
- If $r, r_1, r_2 \in$ **SERE** then each of the following are in the set **SERE** as well:

$\{r\}$	(Braced SERE)
$r_1 ; r_2$	(Concatenation)
$r_1 : r_2$	(Fusion)
$r_1 \mid r_2$	(Or)
$r_1 \,\&\&\, r_2$	(Length-matching And)
$r\,[*0]$	(Empty sequence)
$r\,[*]$	(Consecutive repetition)

Definition 36 (**Foundation Language Formulas - FL** [IEE12a, IEE05, Acc04])**.** The set **FL** is defined as the smallest set that satisfies the following:

- If $b \in$ **BoolExpr** then $b, b\,! \in$ **FL**
- If $\varphi, \varphi_1, \varphi_2 \in$ **FL**, $r \in$ **SERE** and $b \in$ **BoolExpr** then each of the following are in the set **FL** as well:

(φ)	(Parenthesis)
$\neg\varphi$	(Negation)
$\varphi_1 \wedge \varphi_2$	(And)
$\mathsf{X!}\,\varphi$	(Strong Next)
$[\varphi_1 \,\mathsf{U}\, \varphi_2]$	(Strong Until)
$\varphi \,\mathsf{abort}\, b$	(Abort)
r	(SERE)
$r\,!$	(Tight SERE)
$r \mapsto \varphi$	(Suffix implication)

The rest of the operators in the foundation language are defined as syntactic sugar. For easy reference we provide a few commonly used examples:

$b_1 \vee b_2$	$\stackrel{def}{=}$	$\neg(\neg b_1 \wedge \neg b_2)$	
$b_1 \rightarrow b_2$	$\stackrel{def}{=}$	$\neg b_1 \vee b_2$	
$b_1 \leftrightarrow b_2$	$\stackrel{def}{=}$	$(b_1 \rightarrow b_2) \wedge (b_2 \rightarrow b_1)$	
$r[*k]$	$\stackrel{def}{=}$	$\overbrace{r ; r ; \cdots ; r}^{k \text{ times}}$	
$r\,[+]$	$\stackrel{def}{=}$	$r ; r\,[*]$	
$[*]$	$\stackrel{def}{=}$	$\mathsf{true}\,[*]$	
$[+]$	$\stackrel{def}{=}$	$\mathsf{true}\,[+]$	
$r_1 \,\&\, r_2$	$\stackrel{def}{=}$	$\{r_1 \,\&\&\, \{r_2 ; [*]\}\} \mid \{\{r_1 ; [*]\} \,\&\&\, r_2\}$	(Non-length-matching And)
$r_1 \,\mathsf{within}\, r_2$	$\stackrel{def}{=}$	$\{[*] ; r_1 ; [*]\} \,\&\&\, r_2$	(Within)
$\varphi_1 \vee \varphi_2$	$\stackrel{def}{=}$	$\neg(\neg\varphi_1 \wedge \neg\varphi_2)$	(Or)
$\varphi_1 \rightarrow \varphi_2$	$\stackrel{def}{=}$	$\neg\varphi_1 \vee \varphi_2$	(Implication)
$\varphi_1 \leftrightarrow \varphi_2$	$\stackrel{def}{=}$	$(\varphi_1 \rightarrow \varphi_2) \wedge (\varphi_2 \rightarrow \varphi_1)$	(If and only if)
$\mathsf{F}\,\varphi$	$\stackrel{def}{=}$	$[\mathsf{true}\,\mathsf{U}\,\varphi]$	(Strong Eventually)
$\mathsf{G}\,\varphi$	$\stackrel{def}{=}$	$\neg\,\mathsf{F}\,\neg\varphi$	(Always)
$\mathsf{X}\,\varphi$	$\stackrel{def}{=}$	$\neg\,\mathsf{X!}\,\neg\varphi$	(Next)

B.2 Semantics

The meaning of a formula in PSL is defined with respect to a sequence (finite or infinite) $v = v_0v_1v_2...$ over the alphabet $\Sigma = 2^{\mathbf{AP}} \cup \{\bot, \top\}$, this is, each $v_i \in 2^{\mathbf{AP}} \cup \{\bot, \top\}$. If $v_i \in 2^{\mathbf{AP}}$ then v_i is a set of atomic propositions which hold true at that point in the sequence. If $v_i = \top$, this represents that any boolean expression holds at that point in the sequence. If $v_i = \bot$, it represents that nothing is true at that no boolean expression holds at that point. The empty sequence is denoted ϵ. The i-th item of a sequence v is denoted v_{i-1} or v^{i-1} (since the first index is 0). We write $v^{i..j}$ to be the subsequence $v_iv_{i+1}...v_j$ for $j \geqslant i$, and $v^{i\cdot\cdot}$the (finite or infinite) suffix of v starting from index i. We write $|v|$ for the length of v. If $\ell \in \Sigma$, we write ℓ^ω for the infinite sequence $\ell\ell\ell$.... If v is a sequence, we write $\bar{v}$ for the sequence obteined by replacing each $\bot$ with $\top$ and each $\top$ with $\bot$. We write Σ^* for the set of finite sequences over Σ, Σ^ω for the set of infinite sequences over Σ and $\Sigma^\infty \stackrel{def}{=} \Sigma^* \cup \Sigma^\omega$.

Definition 37 (**Semantics of boolean expressions** [IEE12a, IEE05, Acc04])**.** We define $\models \subseteq \Sigma \times \mathbf{BoolExpr_{AP}}$ as the smallest relation satisfying the following (writing $\ell \models b$ for $(\ell, b) \in \models$):

- $\top \models b$ for any $b \in \mathbf{BoolExpr}$
- $\bot \not\models b$ for all $b \in \mathbf{BoolExpr}$
- $\ell \models \mathsf{true}$ where $\ell \in 2^{\mathbf{AP}}$
- $\ell \not\models \mathsf{false}$ where $\ell \in 2^{\mathbf{AP}}$
- $\ell \models p$ iff $p \in \ell$ where $p \in \mathbf{AP}$ and $\ell \in 2^{\mathbf{AP}}$
- $\ell \models \neg b$ iff $\ell \not\models b$ where $\ell \in 2^{\mathbf{AP}}$
- $\ell \models b_1 \wedge b_2$ iff $\ell \models b_1$ and $\ell \models b_2$ where $\ell \in 2^{\mathbf{AP}}$

Definition 38 (**Semantics of SEREs** [IEE12a, IEE05, Acc04])**.** We define $\models \subseteq \Sigma^\infty \times \mathbf{SERE}$ as the smallest relation satisfying the following (writing $v \models r$ for $(v, r) \in \models$):

- $v \models \{r\}$ iff $v \models r$
- $v \models b$ iff $|v| = 1$ and $v \models b$
- $v \models r_1 ; r_2$ iff $\exists u \in \Sigma^*, w \in \Sigma^\infty . v = uw$, $u \models r_1$ and $w \models r_2$
- $v \models r_1 : r_2$ iff $\exists u \in \Sigma^*, \ell \in \Sigma, w \in \Sigma^\infty . v = u\ell w$, $u\ell \models r_1$ and $\ell w \models r_2$
- $v \models r_1 \mid r_2$ iff $v \models r_1$ or $v \models r_2$
- $v \models r_1 \,\&\&\, r_2$ iff $v \models r_1$ and $v \models r_2$
- $v \models r\,[*0]$ iff $v = \epsilon$
- $v \models r\,[*]$ iff $v \models r\,[*0]$ or $\exists u \in \Sigma^*, w \in \Sigma^\infty . u \neq \epsilon$, $v = uw$, $u \models r$ and $w \models r\,[*]$

Definition 39 (**Semantics of FL [IEE12a, IEE05, Acc04]**)**.** We define $\vDash \subseteq \Sigma^\infty \times \mathbf{FL}$ as the smallest relation satisfying the following (writing $v \vDash \varphi$ for $(v, \varphi) \in \vDash$):

- $v \vDash (\varphi)$ iff $v \vDash \varphi$
- $v \vDash \neg\varphi$ iff $\bar{v} \not\vDash \varphi$
- $v \vDash \varphi_1 \wedge \varphi_2$ iff $v \vDash \varphi_1$ and $v \vDash \varphi_2$
- $v \vDash b!$ iff $|v| > 0$ and $v^0 \models b$
- $v \vDash b$ iff $|v| = 0$ or $v^0 \models b$
- $v \vDash r\,!$ iff $\exists j < |v| . v^{0..j} \models r$

- $v \models r$ iff $\forall j < |v|.\, v^{0..j}\top^\omega \models r\,!$
- $v \models \mathsf{X!}\,\varphi$ iff $|v| > 1$ and $v^{1..} \models \varphi$
- $v \models [\varphi_1 \,\mathsf{U}\, \varphi_2]$ iff $\exists k < |v|.\, v^{k..} \models \varphi_2$ and $\forall j < k.\, v^{j..} \models \varphi_1$
- $v \models \varphi \,\mathsf{abort}\, b$ iff $v \models \varphi$ or $\exists j < |v|.\, v^j \Vdash b$ and $v^{0..j-1}\top^\omega \models \varphi$
- $v \models r \mapsto \varphi$ iff $\forall j < |v|.$ if $\bar{v}^{0..j} \models r$ then $v^{j..} \models \varphi$

C Proofs

C.1 Theoretical framework

Proposition 1. *For any given complete specification theory $(\mathcal{S}, \otimes, /, \wedge, \leqslant)$, for any $X, P, P', Q, Q', R \in \mathcal{S}$:*

(i) if $P \wedge Q$ is defined and $X \leqslant P \wedge Q$ then $X \leqslant P$ and $X \leqslant Q$

(ii) if $P \wedge Q$ and $P' \wedge Q'$ are defined and $P' \leqslant P$ and $Q' \leqslant Q$ then $P' \wedge Q' \leqslant P \wedge Q$

(iii) if $P \wedge Q$ and $Q \wedge P$ are defined then $P \wedge Q = Q \wedge P$

(iv) if $P \wedge Q$, $Q \wedge R$, $P \wedge (Q \wedge R)$ and $(P \wedge Q) \wedge R$ are defined then $P \wedge (Q \wedge R) = (P \wedge Q) \wedge R$

(v) $P \wedge P = P$

(vi) if Q/P is defined and $X \leqslant Q/P$ then $P \otimes X \leqslant Q$

(vii) if $X \leqslant Y$ and $P \otimes Y \leqslant Q$ then $P \otimes X \leqslant Q$

Proof.

(i) Suppose that $X \leqslant P \wedge Q$. We know by Axiom **(A6)** that $P \wedge Q \leqslant P$ and $P \wedge Q \leqslant Q$ and so, by transitivity we have $X \leqslant P$ and $X \leqslant Q$.

(ii) Assume that $P' \leqslant P$ and $Q' \leqslant Q$. By Axiom **(A6)** we have that $P' \wedge Q' \leqslant P'$ and $P' \wedge Q' \leqslant Q'$ to by transitivity we have $P' \wedge Q' \leqslant P$ and $P' \wedge Q' \leqslant Q$, which imply $P' \wedge Q' \leqslant P \wedge Q$ by Axiom **(A7)**.

(iii) We know that $P \wedge Q \leqslant P$ and $P \wedge Q \leqslant Q$ by Axiom **(A6)**, which is the same as stating that $P \wedge Q \leqslant Q$ and $P \wedge Q \leqslant P$, so by Axiom **(A7)**, we get $P \wedge Q \leqslant Q \wedge P$. With a symmetric argument we establish that $Q \wedge P \leqslant P \wedge Q$.

(iv) By Axiom **(A6)** we know that (1) $P \wedge (Q \wedge R) \leqslant P$ and (2) $P \wedge (Q \wedge R) \leqslant Q \wedge R$. Also by Axiom **(A6)**, we know that (3) $Q \wedge R \leqslant Q$ and (4) $Q \wedge R \leqslant R$. Hence, (2) and (3) imply by transitivity we know that (5) $P \wedge (Q \wedge R) \leqslant Q$ and similarly, (2) and (4) imply (6) $P \wedge (Q \wedge R) \leqslant R$. Hence, from (1) and (5) we conclude by Axiom **(A7)** that (7) $P \wedge (Q \wedge R) \leqslant P \wedge Q$ and therefore, also by Axiom **(A7)** we obtain $P \wedge (Q \wedge R) \leqslant (P \wedge Q) \wedge R$ from (7) and (6).

(v) $P \wedge P \leqslant P$ by Axiom **(A6)**. Since $P \leqslant P$ by reflexivity, then $P \leqslant P \wedge P$ by Axiom **(A7)**.

(vi) Assume that $X \leqslant Q/P$. Then, since $P \leqslant P$, by Axiom **(A1)** we have $P \otimes X \leqslant P \otimes (Q/P)$, but by Axiom **(A3)** we know that $P \otimes (Q/P) \leqslant Q$ so by transitivity $P \otimes X \leqslant Q$.

(vii) Assume that $X \leqslant Y$ and $P \otimes Y \leqslant Q$. Since $P \leqslant P$, by Axiom **(A1)** we have $P \otimes X \leqslant P \otimes Y$, so by transitivity we have $P \otimes X \leqslant Q$.

□

Proposition 2. *$\preceq$ is a preorder (i.e., a reflexive and transitive relation).*

Proof.

Reflexivity: for any contract C, $\mathsf{impl}[\![C]\!] \subseteq \mathsf{impl}[\![C]\!]$ and $\mathsf{env}[\![C]\!] \subseteq \mathsf{env}[\![C]\!]$. Hence $C \preceq C$.

Transitivity: take any contracts C, C', C'' such that $C \preceq C'$ and $C' \preceq C''$. Hence $\mathsf{impl}[\![C]\!] \subseteq \mathsf{impl}[\![C']\!]$ and $\mathsf{env}[\![C']\!] \subseteq \mathsf{env}[\![C]\!]$ as well as $\mathsf{impl}[\![C']\!] \subseteq \mathsf{impl}[\![C'']\!]$ and $\mathsf{env}[\![C'']\!] \subseteq \mathsf{env}[\![C']\!]$. From this we obtain that $\mathsf{impl}[\![C]\!] \subseteq \mathsf{impl}[\![C'']\!]$ and $\mathsf{env}[\![C'']\!] \subseteq \mathsf{env}[\![C]\!]$, in other words, $C \preceq C''$.

□

Proposition 3. *Let C and C' be any contracts. For any specification I, if $I \in \mathsf{impl}[\![C]\!]$ and $C \preceq C'$ then $I \in \mathsf{impl}[\![C']\!]$.*

Proof. This is immediate from the definition of contract refinement (Definition 6). □

Theorem 1. *([BDH+ 12a]) Let $C = (A, G)$ and $C' = (A', G')$ be to contracts. Then $C' \preceq C$ if and only if $A \leqslant A'$ and $G' \leqslant_A G$.*

Proof. See [BDH+12b]. □

Proposition 4. *Contract equivalence and strong contract equivalence are equivalence relations.*

Proof. By Definition 7, $\equiv$ and $\simeq$ are symmetric, and also $\equiv \subseteq \leqslant$ and $\simeq \subseteq \preceq$, so they are also reflexive and transitive. □

Proposition 5. *For any contracts C, C',*

(i) if $C \equiv C'$ then $C \simeq C'$

(ii) if C and C' are in normal form then $C \equiv C'$ if and only if $C \simeq C'$

Proof. □

(i) Suppose that $C \equiv C'$. Then $A \leqslant A'$, $A' \leqslant A$, $G \leqslant G'$ and $G' \leqslant G$. We show that $C \preceq C'$ and with a symmetric argument we can prove that $C' \preceq C$. To show that $C \preceq C'$ we need to show that (a) $\forall I.\, I \leqslant_A G \Rightarrow I \leqslant_{A'} G'$ and (b) $\forall E.\, E \leqslant A' \Rightarrow E \leqslant A$.

   (a) Take any I such that $I \leqslant_A G$. This is, for any H, if $H \leqslant A$ then $I \otimes H \leqslant G \otimes H$. We need to show that $I \leqslant_{A'} G'$, this is, for any H', $H' \leqslant A'$ implies $I \otimes H' \leqslant G' \otimes H'$. Take any H' such that $H' \leqslant A'$. Since $A' \leqslant A$ we know that $H' \leqslant A$ and since $I \leqslant_A G$, we obtain that $I \otimes H' \leqslant G \otimes H'$. We also know that $G \leqslant G'$ and since $\leqslant$ must preserve composition $\otimes$ (see Definition 1) we have that $G \otimes H' \leqslant G' \otimes H'$. Hence by transitivity we have that $I \otimes H' \leqslant G' \otimes H'$, as required. We conclude that $I \leqslant_{A'} G'$.

   (b) Let E be such that $E \leqslant A'$. We know that $A' \leqslant A$. Hence we obtain that $E \leqslant A$ as required.

(ii) We only need to show that $C \simeq C'$ implies $C \equiv C'$ whenever C and C' are in normal form. If $C \simeq C'$ then $\mathsf{impl}[\![C]\!] = \mathsf{impl}[\![C']\!]$, this is, for all I, $I \leqslant_A G$ iff $I \leqslant_{A'} G'$. But this is the same as saying that $I \leqslant G$ iff $I \leqslant G'$ for all I, since C and C' are in normal form Definition 8. But we know that $G \leqslant G$, so this implies that $G \leqslant G'$, and similarly, we know that $G' \leqslant G'$ and so $G' \leqslant G$. Now, $C \simeq C'$ also implies that $\mathsf{env}[\![C]\!] = \mathsf{env}[\![C']\!]$. This is, for all E, $E \leqslant A$ iff $E \leqslant A'$. But $A \leqslant A$ and so, $A \leqslant A'$. Similarly, $A' \leqslant A'$ and so $A' \leqslant A$.

Theorem 2. *([BDH+ 12a]) If contracts C_1 and C_2 are dominatible then $C_1 \boxtimes C_2$ is (up to semantic equivalence) the composition of C_1 and C_2.*

Proof. See [BDH+12b]. □

Theorem 3. *[Theorem 6 of [BDH+ 12a]] Let C_1, C_2, D_1, D_2 be contracts with normal forms $\underline{C_1}$, $\underline{C_2}$, $\underline{D_1}$ and $\underline{D_2}$, and such that C_1 and C_2 are dominatible. If $D_1 \preceq C_1$ and $D_2 \preceq C_2$ then $\underline{D_1} \boxtimes \underline{D_2} \preceq \underline{C_1} \boxtimes \underline{C_2}$.*

Proof. See [BDH+12b]. □

Corollary 1. *Let C_1, C_2, D_1, D_2 be contracts with normal forms $\underline{C_1}$, $\underline{C_2}$, $\underline{D_1}$ and $\underline{D_2}$, and such that C_1 and C_2 are dominatible. If $D_1 \simeq C_1$ and $D_2 \simeq C_2$ then $\underline{D_1} \boxtimes \underline{D_2} \simeq \underline{C_1} \boxtimes \underline{C_2}$.*

Proof. This is a direct consequence of Definition 7 and Theorem 3. □

Proposition 6. *Let $(\mathcal{S}, \otimes, /, \wedge, \leqslant)$ be a standard complete specification theory. Then, for all $X, P, Q \in \mathcal{S}$:*

(i) *if* $X \leqslant P \wedge Q$ *then* $X \leqslant P \otimes Q$

(ii) *if* $X \leqslant P$ *and* $X \leqslant Q/P$ *then* $X \leqslant Q$

(iii) $P \otimes Q \leqslant P$

(iv) *if* $P \leqslant Q$ *then* $P/R \leqslant Q/R$

(v) *if* $P \leqslant Q$ *then* $R/Q \leqslant R/P$

(vi) $(P/Q) \otimes R \leqslant (P \otimes R)/Q$

(vii) $P \wedge Q \leqslant Q/P$

Proof. Take any $X, P, Q \in \mathcal{S}$.

(i) Assume that $X \leqslant P \wedge Q$. By Proposition 1 we know that $X \leqslant P$ and $X \leqslant Q$. Then, by Axiom **(A10)** we get $X \leqslant P \otimes Q$.

(ii) Assume that $X \leqslant P$ and $X \leqslant Q/P$. Then by Axiom **(A7)** we know that $X \leqslant P \wedge (Q/P)$. So, by item (i), we know that $X \leqslant P \otimes (Q/P)$. But Axiom **(A3)** states that $P \otimes (Q/P) \leqslant Q$, so by transitivity of $\leqslant$ we get $X \leqslant Q$.

(iii) By reflexivity we know that $P \leqslant P$, so by Axiom **(A11)** we get that $P \otimes Q \leqslant P$.

(iv) Assume that $P \leqslant Q$. We know that $R \otimes (P/R) \leqslant P$ by Axiom **(A3)**, so by transitivity, $R \otimes (P/R) \leqslant Q$, which, by Proposition 1 implies $P/R \leqslant Q/R$.

(v) Assume that $P \leqslant Q$. Since $R/Q \leqslant R/Q$ we have that $P \otimes (R/Q) \leqslant Q \otimes (R/Q)$ by Axiom **(A1)**. But we know that $Q \otimes (R/Q) \leqslant R$ by Axiom **(A3)**, so by transitivity $P \otimes (R/Q) \leqslant R$ which, by Proposition 1 implies $R/Q \leqslant R/P$.

(vi) By Axiom **(A3)**, $Q \otimes (P/Q) \leqslant P$ which by Axiom **(A1)** entails $(Q \otimes (P/Q)) \otimes R \leqslant P \otimes R$. By associativity (Axiom **(A9)**), $Q \otimes ((P/Q) \otimes R) \leqslant P \otimes R$, and so, by Axiom **(A4)**, $(P/Q) \otimes R \leqslant (P \otimes R)/Q$.

(vii) By Axiom **(A6)**, $P \wedge Q \leqslant Q$, so by Axiom **(A11)**, $(P \wedge Q) \otimes P \leqslant Q$, hence by Axiom **(A4)**, $P \wedge Q \leqslant Q/P$.

□

Proposition 7. *Let* $(\mathcal{S}, \otimes, /, \wedge, \leqslant)$ *be a simplified specification theory. Then, for all* $X, P, Q, R \in \mathcal{S}$*:*

(i) $P \wedge Q = P \otimes Q$

(ii) $P \otimes Q \leqslant P$ *and* $P \otimes Q \leqslant Q$

(iii) $(R/Q)/P = R/(P \wedge Q)$

(iv) $(P \wedge Q)/R = (P/R) \wedge (Q/R)$

(v) $(Q/P)/P = Q/P$

(vi) $(P/Q) \otimes (Q/R) \leqslant P/R$

(vii) $(P/Q) \otimes (P/(Q \wedge R)) \leqslant P/(Q \wedge R)$

(viii) $(P/Q) \otimes (P'/Q') \leqslant (P \otimes P')/(Q \otimes Q')$

Proof. Take any $X, P, Q, R \in \mathcal{S}$.

(i) We need to show that $P \wedge Q \leqslant P \otimes Q$ and $P \otimes Q \leqslant P \wedge Q$:

- By reflexivity, $P \wedge Q \leqslant P \wedge Q$ and by (i), $P \wedge Q \leqslant P \otimes Q$.
- We know, by reflexivity we know that $P \leqslant P$, so by Axiom **(A11)**, that (1) $P \otimes Q \leqslant P$. Similarly, by reflexivity we know that $Q \leqslant Q$ and by Axiom **(A11)**, that $Q \otimes P \leqslant Q$. By commutativity we have that $P \otimes Q \leqslant Q \otimes P$ and therefore (2) $P \otimes Q \leqslant Q$. Hence, (1) and (2) imply by Axiom **(A7)** that $P \otimes Q \leqslant P \wedge Q$.

(ii) This is a direct consequence of Item 1 of this Proposition.

(iii) We need to show that $(R/Q)/P \leqslant R/(P \wedge Q)$ and $R/(P \wedge Q) \leqslant (R/Q)/P$:

- Let us call $S \stackrel{def}{=} (R/Q)/P$. By reflexivity we have (1) $S \leqslant S$. By Axiom **(A11)** we get (2) $S \otimes (P \wedge Q) \leqslant S$ from (1). By Axiom **(A8)** we have that (3) $(P \wedge Q) \otimes S \leqslant S \otimes (P \wedge Q)$, so by transitivity we get (4) $(P \wedge Q) \otimes S \leqslant S$ from (2) and (3). Now, by Axiom **(A6)** we know that (5) $P \wedge Q \leqslant P$, which by Axiom **(A11)** implies (6) $(P \wedge Q) \otimes S \leqslant P$. Then from (4) and (6) we obtain that (7) $(P \wedge Q) \otimes S \leqslant R/Q$ by Item 2 of Proposition 6. We also know that (8) $P \wedge Q \leqslant Q$ by Axiom **(A6)**, which again by Axiom **(A11)** implies (9) $(P \wedge Q) \otimes S \leqslant Q$. Then, from (7) and (9) we get that $(P \wedge Q) \otimes S \leqslant R$ by Item 2 of Proposition 6. Therefore, by Axiom **(A4)** we get $S \leqslant R/(P \wedge Q)$.
- Let us call $T \stackrel{def}{=} R/(P \wedge Q)$. By reflexivity we know that (1) $Q \leqslant Q$ and so by Axiom **(A11)** we know that (2) $Q \otimes (P \otimes T) \leqslant Q$. By associativity we know that (3) $(Q \otimes P) \otimes T \leqslant Q \otimes (P \otimes T)$ and so, by transitivity we have that (4) $(Q \otimes P) \otimes T \leqslant Q$ from (3) and (2). Similarly, by reflexivity we know that (5) $P \leqslant P$ and therefore, by Axiom **(A11)** we know that (6) $P \otimes (Q \otimes T) \leqslant P$. By commutativity (Axiom **(A8)**) we know that (7) $Q \otimes P \leqslant P \otimes Q$ and therefore, by Axiom **(A1)** we know that (8) $(Q \otimes P) \otimes T \leqslant (P \otimes Q) \otimes T$ and by associativity, (9) $(P \otimes Q) \otimes T \leqslant P \otimes (Q \otimes T)$. So we get that (10) $(Q \otimes P) \otimes T \leqslant P$ by transitivity from (8), (9), and (6). Hence, by Axiom **(A7)** we conclude that (11) $(Q \otimes P) \otimes T \leqslant P \wedge Q$ from (10) and (4). So again, by reflexivity we know that (12) $T \leqslant T$, which by Axiom **(A11)** entails (13) $T \otimes (Q \otimes P) \leqslant T$. But by commutativity, (14) $(Q \otimes P) \otimes T \leqslant T \otimes (Q \otimes P)$, so by transitivity, (15) $(Q \otimes P) \otimes T \leqslant T$. Then, (11) and (15) imply that (16) $(Q \otimes P) \otimes T \leqslant R$ by Item 2 of Proposition 6. This means, by associativity and transitivity that (17) $Q \otimes (P \otimes T) \leqslant R$. Hence by Axiom **(A4)** (18) $P \otimes T \leqslant R/Q$ and also by Axiom **(A4)** (19) $T \leqslant (R/Q)/P$.

(iv) We need to show that $(P \wedge Q)/R \leqslant (P/R) \wedge (Q/R)$ and $(P/R) \wedge (Q/R) \leqslant (P \wedge Q)/R$

- Let us call $S \stackrel{def}{=} (P \wedge Q)/R$. By Axiom **(A3)** we know that (1) $R \otimes S \leqslant P \wedge Q$. By commutativity (Axiom **(A8)**) we know that (2) $S \otimes R \leqslant R \otimes S$ and therefore, by transitivity, (3) $S \otimes R \leqslant P \wedge Q$. Hence, by Proposition 1 we have (4) $S \otimes R \leqslant P$ and (5) $S \otimes R \leqslant Q$. So by Axiom **(A4)** we obtain (6) $S \leqslant P/R$ and (7) $S \leqslant Q/R$, which together imply $S \leqslant (P/R) \wedge (Q/R)$ by Axiom **(A7)**.
- Let us call $T \stackrel{def}{=} (P/R) \wedge (Q/R)$. By Axiom **(A6)** we know that (1) $T \leqslant P/R$ and (2) $T \leqslant Q/R$. We also know that (3) $R \leqslant R$ by reflexivity, so from (3) and (1) we obtain (4) $R \otimes T \leqslant P$ by Item 2 of Proposition 6, and similarly from (3) and (2) we obtain (5) $R \otimes T \leqslant Q$. Hence, from (4) and (5) we obtain (6) $R \otimes T \leqslant P \wedge Q$ by Axiom **(A7)**, which entails $T \leqslant (P \wedge Q)/R$ by Axiom **(A4)**.

(v) By Proposition 7(Item 3) we know that $(Q/P)/P = Q/(P \wedge P)$. By Proposition 1(Item 5), $P \wedge P = P$, hence, by Proposition 6(Item 5), $Q/(P \wedge P) = Q/P$, so by transitivity, $(Q/P)/P = Q/P$.

(vi) We derive this as follows:

$$
\begin{array}{lcll}
R \otimes ((P/Q) \otimes (Q/R)) & = & (R \otimes (P/Q)) \otimes (Q/R) & \text{by Axiom } \mathbf{(A9)} \\
& = & ((P/Q) \otimes R) \otimes (Q/R) & \text{by Axiom } \mathbf{(A8)} \text{ and Axiom } \mathbf{(A1)} \\
& = & (P/Q) \otimes (R \otimes (Q/R)) & \text{by Axiom } \mathbf{(A9)} \\
& \leqslant & (P/Q) \otimes Q & \text{by Axiom } \mathbf{(A3)} \text{ and Axiom } \mathbf{(A1)} \\
& = & Q \otimes (P/Q) & \text{by Axiom } \mathbf{(A8)} \\
& \leqslant & P & \text{by Axiom } \mathbf{(A3)}
\end{array}
$$

Hence we have that $R \otimes ((P/Q) \otimes (Q/R)) \leqslant P$, so by Axiom **(A4)** we get $(P/Q) \otimes (Q/R) \leqslant P/R$.

(vii) We can derive this as follows:

$$
\begin{array}{lcll}
(P/Q) \otimes (P/(Q \wedge R)) & = & (P/Q) \otimes ((P/R)/Q) & \text{by Proposition 7(Item 3) and Axiom } \mathbf{(A1)} \\
& = & (P/Q) \wedge ((P/R)/Q) & \text{by Proposition 7(Item 1)} \\
& = & (P \wedge (P/R))/Q & \text{by Proposition 7(Item 4)} \\
& = & (P \otimes (P/R))/Q & \text{by Proposition 7(Item 3) and Proposition 6(Item 4)} \\
& = & (P \wedge (P/R))/Q & \text{by Proposition 7(Item 1) and Proposition 6(Item 4)} \\
& \leqslant & ((P \wedge P)/R)/Q & \text{by Proposition 6(Item 6)} \\
& = & (P/R)/Q & \text{by Proposition 1(Item 5) and Proposition 6(Item 4)} \\
& = & P/(Q \wedge R) & \text{by Proposition 7(Item 3)}
\end{array}
$$

(viii) We obtain $(P/Q) \otimes (P'/Q') \leqslant (P \otimes P')/(Q \otimes Q')$ as follows: let $\Phi \stackrel{def}{=} (P/Q) \otimes (P'/Q')$

1	$\Phi \leqslant P/Q$	by Axiom **(A6)**
2	$\Phi \otimes Q \leqslant P/Q$	by Axiom **(A11)** from 1
3	$\Phi \otimes Q \leqslant Q$	by Proposition 7(Item 2)
4	$\Phi \otimes Q \leqslant P$	by Proposition 6(Item 2) from 2,3
5	$\Phi \leqslant P'/Q'$	by Axiom **(A6)**
6	$\Phi \otimes Q' \leqslant P'/Q'$	by Axiom **(A11)** from 5
7	$\Phi \otimes Q' \leqslant Q'$	by Proposition 7(Item 2)
8	$\Phi \otimes Q' \leqslant P'$	by Proposition 6(Item 2) from 6,7
9	$(\Phi \otimes Q) \otimes Q' \leqslant P$	by Axiom **(A11)** from 4
10	$\Phi \otimes (Q \otimes Q') \leqslant P$	by Axiom **(A9)**
11	$(\Phi \otimes Q') \otimes Q \leqslant P'$	by Axiom **(A11)** from 8
12	$\Phi \otimes (Q' \otimes Q) \leqslant P'$	by Axiom **(A9)**
13	$\Phi \otimes (Q \otimes Q') \leqslant P'$	by Axiom **(A8)**
14	$\Phi \otimes (Q \otimes Q') \leqslant P \otimes P'$	by Axiom **(A10)**
15	$(Q \otimes Q') \otimes \Phi \leqslant P \otimes P'$	by Axiom **(A8)**
16	$\Phi \leqslant (P \otimes P')/(Q \otimes Q')$	by Axiom **(A4)**

□

Lemma 1. *Let $(\mathcal{S}, \otimes, /, \wedge, \leqslant)$ be a simplified specification theory. Then, for all $P, Q, R \in \mathcal{S}$. Then $P \leqslant_R Q$ if and only if $P \otimes R \leqslant Q$.*

Proof. ($\Rightarrow$) Assume that $P \leqslant_R Q$. This is, for all R' such that $R' \leqslant R$, $P \otimes R' \leqslant Q \otimes R'$. By reflexivity we know that $R \leqslant R$ and therefore $P \otimes R \leqslant Q \otimes R$. But by Proposition 6(Item 3) we have that $Q \otimes R \leqslant Q$, so by transitivity, $P \otimes R \leqslant Q$.

($\Leftarrow$) Assume that $P \otimes R \leqslant Q$. Suppose that $R' \leqslant R$. Then, by Axiom **(A1)** we have that $P \otimes R' \leqslant P \otimes R$, and so, by transitivity we obtain (1) $P \otimes R' \leqslant Q$. And by Proposition 7(Item 2) we know that (2) $P \otimes R' \leqslant R'$. From (1) and (2) we can infer that $P \otimes R' \leqslant Q \otimes R'$ by Axiom **(A10)**. Hence we have shown that for any $R' \leqslant R$, $P \otimes R' \leqslant Q \otimes R'$, this is, $P \leqslant_R Q$. □

Proposition 8. *Let $(\mathcal{S}, \otimes, /, \wedge, \leqslant)$ be a simplified specification theory and let $C = (A, G)$. Then*

(i) $C \simeq \underline{C}$

(ii) $\underline{C}$ *is in normal form.*

(iii) $\underline{\underline{C}} \equiv \underline{C}$

Proof. Let $C = (A, G)$. Then, by Definition 15, $\underline{C} = (A, \underline{G})$ with $\underline{G} = G/A$.

(i) We have to prove that (a) $\mathsf{impl}[\![C]\!] = \mathsf{impl}[\![\underline{C}]\!]$ and (b) $\mathsf{env}[\![C]\!] = \mathsf{env}[\![\underline{C}]\!]$.

(a) To prove that $\mathsf{impl}[\![C]\!] = \mathsf{impl}[\![\underline{C}]\!]$ we need to show that for all I, $I \leqslant_A G$ if and only if $I \leqslant_A \underline{G}$. But, by the characterization of relativized refinement given by Lemma 1, this is the same as showing that for all I, $I \otimes A \leqslant G$ if and only if $I \otimes A \leqslant \underline{G}$. We establish this as follows:

($\Rightarrow$) Assume that $I \otimes A \leqslant G$. We know by Proposition 7(Item 2) that $A \otimes G \leqslant G$. Therefore, by Axiom **(A4)**, $G \leqslant G/A$. Thus, by transitivity, $I \otimes A \leqslant G/A$, *i.e.*, $I \otimes A \leqslant \underline{G}$.

($\Leftarrow$) Assume that $I \otimes A \leqslant \underline{G}$, this is, $I \otimes A \leqslant G/A$. We know by Proposition 7(Item 2) that $I \otimes A \leqslant A$. Therefore, by Proposition 6(Item 2), $I \otimes A \leqslant G$.

(b) Since $\mathsf{assumption}(C) = \mathsf{assumption}(\underline{C})$ then $\mathsf{env}[\![C]\!] = \{E \mid E \leqslant A\} = \mathsf{env}[\![\underline{C}]\!]$.

(ii) We have to prove that $I \leqslant_A \underline{G}$ if and only if $I \leqslant \underline{G}$, but by Lemma 1, this is the same as proving that $I \otimes A \leqslant \underline{G}$ if and only if $I \leqslant \underline{G}$.

($\Rightarrow$) Assume that $I \otimes A \leqslant \underline{G}$, *i.e.*, $I \otimes A \leqslant G/A$. We know, by Proposition 7(Item 2) , that $I \otimes A \leqslant A$. Then, by Proposition 6(Item 2), we have that $I \otimes A \leqslant G$, but this implies, by Axiom **(A4)**, that $I \leqslant G/A$, *i.e.*, $I \leqslant \underline{G}$.

($\Leftarrow$) Assume that $I \leqslant \underline{G}$, *i.e.*, $I \leqslant G/A$. By Proposition 1(Item 6) we have that $A \otimes I \leqslant G$, and by commutativity (Axiom **(A8)**) we know that $I \otimes A \leqslant A \otimes I$, so by transitivity, $I \otimes A \leqslant G$. But $A \otimes G \leqslant G$ by Proposition 7(Item 2), and so, by Axiom **(A4)**, $G \leqslant G/A$. Therefore, by transitivity, $I \otimes A \leqslant G/A$, which is to say $I \otimes A \leqslant \underline{G}$.

(iii) Since $\underline{C} = (A, G/A)$ then $\underline{\underline{C}} = (A, (G/A)/A)$, but by Proposition 7(Item 5), $(G/A)/A = G/A$. Therefore $\underline{C} = (A, G/A) \equiv (A, (G/A)/A) = \underline{\underline{C}}$.

□

C.2 Specifications and contracts

Theorem 4. *PSL is a simplified specification theory* $(\mathbf{PSL}, \otimes^{\mathsf{psl}}, /^{\mathsf{psl}}, \wedge^{\mathsf{psl}}, \leqslant^{\mathsf{psl}})$ *where:*

- $\mathcal{S} \stackrel{def}{=} \mathbf{PSL}$ *is the set of PSL expressions*
- *Composition* $\otimes^{\mathsf{psl}}$ *is PSL conjunction:* $\varphi_1 \otimes^{\mathsf{psl}} \varphi_2 \stackrel{def}{=} \varphi_1 \wedge \varphi_2$
- *Quotient* $/^{\mathsf{psl}}$ *is PSL implication:* $\varphi_1 /^{\mathsf{psl}} \varphi_2 \stackrel{def}{=} \varphi_2 \rightarrow \varphi_1$
- *Conjunction* $\wedge^{\mathsf{psl}}$ *is PSL conjunction:* $\varphi_1 \wedge^{\mathsf{psl}} \varphi_2 \stackrel{def}{=} \varphi_1 \wedge \varphi_2$ *(where the right-hand side represents the PSL conjunction operator)*
- *Refinement is logical entailment:* $\varphi_1 \leqslant^{\mathsf{psl}} \varphi_2$ *iff* $\vDash \varphi_1 \rightarrow \varphi_2$

Proof. We need to show that $\leqslant^{\mathsf{psl}}$ is a preorder and that axioms in Definition 1, Definition 12 and Definition 13. That $\leqslant^{\mathsf{psl}}$ is a preorder follows from the fact that $\rightarrow$ is reflexive and transitive.

(A1) Take any $P', P, Q', Q \in \mathbf{PSL}$ such that $P' \leqslant^{\mathsf{psl}} P$ and $Q' \leqslant^{\mathsf{psl}} Q$. This is, $\vDash P' \rightarrow P$ and $\vDash Q' \rightarrow Q$. Hence, for any sequence v, $v \vDash P' \rightarrow P$ and $v \vDash Q' \rightarrow Q$, or in other words, $v \vDash P'$ implies $v \vDash P$ and $v \vDash Q'$ implies $v \vDash Q$. Take any sequence u. Assume that $u \vDash P' \otimes^{\mathsf{psl}} Q'$, this is, $u \vDash P' \wedge Q'$. Then, by the definition of $\vDash$, $u \vDash P'$ and $u \vDash Q'$. Hence, $u \vDash P$ and $u \vDash Q$ and therefore $u \vDash P \wedge Q$, *i.e.*, $u \vDash P \otimes^{\mathsf{psl}} Q$. So for any u, $u \vDash P' \otimes^{\mathsf{psl}} Q'$ implies $u \vDash P \otimes^{\mathsf{psl}} Q$, which means that $u \vDash P' \otimes^{\mathsf{psl}} Q' \rightarrow P \otimes^{\mathsf{psl}} Q$ for any u. This is, $\vDash P' \otimes^{\mathsf{psl}} Q' \rightarrow P \otimes^{\mathsf{psl}} Q$, which is to say that $P' \otimes^{\mathsf{psl}} Q' \leqslant^{\mathsf{psl}} P \otimes^{\mathsf{psl}} Q$.

(A2) We have to prove that for any $P, Q \in \mathbf{PSL}$, $Q \,/^{\mathsf{psl}}\, P$ is defined if and only if there is an $X \in \mathbf{PSL}$ such that $P \otimes^{\mathsf{psl}} X \leqslant^{\mathsf{psl}} Q$. In other words we need to show that $P \to Q$ is defined iff there is an X such that $\vDash P \wedge X \to Q$. This is so because for any formulas $P, Q \in \mathbf{PSL}$, $P \to Q \in \mathbf{PSL}$

(A3) We have to show that for any $P, Q \in \mathbf{PSL}$, $P \otimes^{\mathsf{psl}} (Q \,/^{\mathsf{psl}}\, P) \leqslant^{\mathsf{psl}} Q$, *i.e.*, that $\vDash P \wedge (P \to Q) \to Q$. This follows directly from modus ponens (a.k.a. implication elimination) which holds for PSL as it holds for classical propositional logic.

(A4) We have to show that for any $P, Q, X \in \mathbf{PSL}$, if $P \otimes^{\mathsf{psl}} X \leqslant^{\mathsf{psl}} Q$ then $X \leqslant^{\mathsf{psl}} Q \,/^{\mathsf{psl}}\, P$. This is, we need to show that $\vDash P \wedge X \to Q$ implies $\vDash X \to (P \to Q)$. This holds for PSL as it holds for classical propositional logic.

(A5) $P \wedge^{\mathsf{psl}} Q$ is always defined since for any formulas $P, Q \in \mathbf{PSL}$, $P \wedge Q$ is always defined.

(A6) This follows from the fact that for any $P, Q \in \mathbf{PSL}$, $\vDash P \wedge Q \to P$ (*i.e.*, $P \wedge^{\mathsf{psl}} Q \leqslant^{\mathsf{psl}} P$) and $\vDash P \wedge Q \to Q$ (*i.e.*, $P \wedge^{\mathsf{psl}} Q \leqslant^{\mathsf{psl}} Q$) hold for PSL as it holds for classical propositional logic by conjunction elimination.

(A7) This follows from the fact that for any $P, Q, X \in \mathbf{PSL}$, if $\vDash X \to P$ (*i.e.*, $X \leqslant^{\mathsf{psl}} P$) and $\vDash X \to Q$ (*i.e.*, $X \leqslant^{\mathsf{psl}} Q$) we have that $\vDash X \to P \wedge Q$, i.e. $X \leqslant^{\mathsf{psl}} P \wedge^{\mathsf{psl}} Q$. This holds in PSL as it holds for classical propositional logic using conjunction introduction.

That this complete specification theory is a strongly commutative monoid (Definition 12) follows directly from the commutativity and associativity of logical conjunction in PSL. That it is a standard specification theory (Definition 13) follows from the fact that composition and conjunction are the same operator, namely, PSL conjunction. □

C.3 Conformance

Theorem 5. *Given an interface F annotated with a contract $C = (A, G)$, and a protocol R annotated with a specification S, F conforms to R if and only if for all ports $p \in \mathsf{ports}(F)$:*

1. $\vDash G \to \mathsf{pproj}[\![S]\!]_{R|F.p}$, *and*
2. $\vDash \mathsf{flip}[\![A]\!] \to \mathsf{pproj}[\![S]\!]_{R|F.p}$

Proof. This follows from Definition 22 and the fact that for any PSL formulas φ_1 and φ_2, $\vDash \varphi_1 \to \varphi_2$ iff for all v, $v \vDash \varphi_1$ implies $v \vDash \varphi_2$, this is, for all v, $v \in \mathcal{L}_{\mathbf{PSL}}(\varphi_1)$ implies $v \in \mathcal{L}_{\mathbf{PSL}}(\varphi_2)$, in other words, $\mathcal{L}_{\mathbf{PSL}}(\varphi_1) \subseteq \mathcal{L}_{\mathbf{PSL}}(\varphi_2)$. □

C.4 Compositional inference

Most of the proofs of this section can be derived directly from the statements in Section 3 and the characterization of PSL as a simplified specification theory (Theorem 4), but we provide explicit versions of the proofs.

Proposition 9. *[**Relativized refinement in PSL**] Let $P, Q, R \in \mathbf{FL}$. $P \leqslant_{\mathrm{R}}^{\mathrm{psl}} Q$ iff for all R' such that $\vDash R' \to R$, $\vDash P \wedge R' \to Q \wedge R'$.*

Proof. This follows directly from Definition 4 and the definitions of composition and refinement from Theorem 4. □

Lemma 2. *Let $P, Q, R \in \mathbf{FL}$. Then $P \leqslant_{\mathrm{R}}^{\mathrm{psl}} Q$ iff $\vDash P \wedge R \to Q$.*

Proof. ($\Rightarrow$) Assume that $P \leqslant_{\mathrm{R}}^{\mathsf{psl}} Q$, this is, for any R' such that $\vDash R' \rightarrow R$, $\vDash P \wedge R' \rightarrow Q \wedge R'$. Let u be any sequence such that $u \vDash P \wedge R$. We now that $R \leqslant^{\mathsf{psl}} R$ by reflexivity, *i.e.*, $\vDash R \rightarrow R$, and so, by our assumption that $P \leqslant_{\mathrm{R}}^{\mathsf{psl}} Q$, we have that $\vDash P \wedge R \rightarrow Q \wedge R$. And since this holds for all sequences, it holds for u: $u \vDash P \wedge R \rightarrow Q \wedge R$, which means that $u \vDash P \wedge R$ implies $u \vDash Q \wedge R$. And since we assumed $u \vDash P \wedge R$, we obtain $u \vDash Q \wedge R$. This entails that $u \vDash Q$ by the definition of $\vDash$. Hence we have shown that $u \vDash P \wedge R \rightarrow Q$ for any u, this is, $\vDash P \wedge R \rightarrow Q$.

($\Leftarrow$) Assume that $\vDash P \wedge R \rightarrow Q$. So for any sequence v, $v \vDash P \wedge R \rightarrow Q$, this is, $v \vDash P \wedge R$ implies $v \vDash Q$. Take any R' such that $R' \leqslant^{\mathsf{psl}} R$, this is, $\vDash R' \rightarrow R$. Let u be any sequence such that $u \vDash R' \rightarrow R$. So $u \vDash R'$ implies $u \vDash R$. Since $\vDash P \wedge R \rightarrow Q$, we know that $u \vDash P \wedge R$ implies $u \vDash Q$. Assume that $u \vDash P \wedge R'$. Hence $u \vDash P$ and $u \vDash R'$, which entails that $u \vDash R$. Since we now know that both $u \vDash P$ and $u \vDash R$ hold, we have that $u \vDash P \wedge R$ holds, which entails $u \vDash Q$. And since we have $u \vDash Q$ and $u \vDash R'$, we have $u \vDash Q \wedge R'$. So from $u \vDash P \wedge R'$ we have obtained $u \vDash Q \wedge R'$, this is, we have proven that $u \vDash P \wedge R' \rightarrow Q \wedge R'$ for any u, *i.e.*, $\vDash P \wedge R' \rightarrow Q \wedge R'$ for any R' such that $\vDash R' \rightarrow R$ which is to say that $P \leqslant_{\mathrm{R}}^{\mathsf{psl}} Q$. □

Proposition 10. *[**Contract implementations and environments in PSL**] Let $C = (A, G)$ be a PSL contract. By*

$$\mathsf{impl}[\![C]\!] = \{I \in \mathbf{FL} \mid \vDash I \wedge A \rightarrow G\}$$

and

$$\mathsf{env}[\![C]\!] = \{E \in \mathbf{FL} \mid \vDash E \rightarrow A\}$$

Proof. This follows directly from Definition 5, the definition of refinement from Theorem 4 and the characterization in PSL of relativized refinement from Lemma 2. □

Proposition 11. *Let $C = (A, G)$ and $C' = (A', G')$ be PSL contracts. $C' \preceq C$ if for all implementations $I \in \mathbf{FL}$, $\vDash I \wedge A' \rightarrow G'$ implies $\vDash I \wedge A \rightarrow G$ (see Lemma 2) and for all environments $E \in \mathbf{PSL}$, $\vDash E \rightarrow A$ implies $\vDash E \rightarrow A'$.*

Proof. This follows directly from the definition of contract refinement Definition 6 and Proposition 10. □

Proposition 12. *Let $C = (A, G)$ and $C' = (A', G')$ be PSL contracts. We have that $C \equiv C'$ iff $\vDash A \leftrightarrow A'$ and $\vDash G \leftrightarrow G'$. And $C \simeq C'$ iff (1) for all I, $\vDash I \wedge A \rightarrow G$ iff $\vDash I \wedge A' \rightarrow G'$, and (2) for all E, $\vDash E \rightarrow A$ iff $\vDash E \rightarrow A'$.*

Proof. That $C \equiv C'$ iff $\vDash A \leftrightarrow A'$ and $\vDash G \leftrightarrow G'$ follows from the definition of strong semantic equivalence (Definition 7) and the definition of refinement in PSL (Theorem 4). That $C \simeq C'$ iff (1) and (2), follows from the definition of semantic equivalence (Definition 7), and Proposition 10. □

Proposition 13. *Let $C = (A, G)$ be a PSL contract. C is in normal form if $\vDash I \wedge A \rightarrow G$ iff $\vDash I \rightarrow G$.*

Proof. This comes from the definition of normal form (Definition 8), the characterization of relativized refinement in PSL (Lemma 2) and the definition of refinement from Theorem 4. □

Proposition 14. *Given a PSL contract $C = (A, G)$:*

(i) $C \simeq \underline{C}$

(ii) $\underline{C}$ is in normal form.

Proof.

(i) Since $C = (A, G)$ and $\underline{C} = (A, \underline{G})$ have the same assumptions, we only need to prove that for all I, $\vDash I \wedge A \rightarrow G$ if and only if $\vDash I \wedge A \rightarrow \underline{G}$.

(a) ($\Rightarrow$) Assume that $\vDash I \wedge A \rightarrow G$. Now, suppose that $\vDash I \wedge A$. Hence we conclude that $\vDash G$ which can be weakened to $\vDash A \rightarrow G$, which is the same as $\vDash \underline{G}$. Therefore $\vDash I \wedge A \rightarrow \underline{G}$.

(b) ($\Leftarrow$) Assume that $\models I \wedge A \rightarrow \underline{G}$. Now, suppose that $\models I \wedge A$. Hence we conclude that $\models \underline{G}$ which is the same as $\models A \rightarrow G$. Since we assumed that $\models I \wedge A$ we have that $\models A$ which, with $\models A \rightarrow G$ allows us to conclude that $\models G$. Therefore $\models I \wedge A \rightarrow G$.

(ii) We only need to prove that for all I, $\models I \wedge A \rightarrow \underline{G}$ if and only if $\models I \rightarrow \underline{G}$.

(a) ($\Rightarrow$) Assume that $\models I \wedge A \rightarrow \underline{G}$. Since $C \simeq \underline{C}$, $\models I \wedge A \rightarrow G$ iff $\models I \wedge A \rightarrow \underline{G}$ and therefore we know that $\models I \wedge A \rightarrow G$. By the rules of propositional logic this is equivalent to $\models I \rightarrow (A \rightarrow G)$ which is $\models I \rightarrow \underline{G}$.

(b) ($\Leftarrow$) Assume that $\models I \rightarrow \underline{G}$. This is, $\models I \rightarrow (A \rightarrow G)$. By the rules of propositional logic this is equivalent to $\models I \wedge A \rightarrow G$, and since $C \simeq \underline{C}$, this is equivalent to $\models I \wedge A \rightarrow \underline{G}$.

□

Corollary 2. *For all $I \in \mathbf{FL}$, $I \in \mathsf{impl}[\![C]\!]$ if and only if $\models I \rightarrow (A \rightarrow G)$*

Proof. A direct proof follows from the definition of $\mathsf{impl}[\![C]\!]$, which implies that $\models I \wedge A \rightarrow G$, which is logically equivalent to $\models I \rightarrow (A \rightarrow G)$. Alternatively, by Proposition 14, $\mathsf{impl}[\![C]\!] = \mathsf{impl}[\![\underline{C}]\!]$ which means that $I \leqslant_A G$ iff $I \leqslant \underline{G}$, or in other words, $\models I \wedge A \rightarrow G$ iff $\models I \rightarrow \underline{G}$. □

Corollary 3. *Let $C = (A, G)$ and $C' = (A', G')$ be to PSL contracts. Then $C' \preceq C$ if and only if $\models A \rightarrow A'$ and $\models G' \rightarrow (A \rightarrow G)$.*

Proof. Since $C \simeq \underline{C}$ then $C' \preceq C$ is equivalent to $C' \preceq \underline{C}$, which by Theorem 1 is the same as saying that $A \leqslant^{\mathsf{psl}} A'$ (which is $\models A \rightarrow A'$) and $G' \leqslant^{\mathsf{psl}}_{\mathrm{A}} \underline{G}$. And this is equivalent to $G' \leqslant^{\mathsf{psl}} \underline{G}$ since $\underline{C}$ is in normal form (Proposition 14) and this in turn is equivalent to $\models G' \rightarrow \underline{G}$, *i.e.*, $\models G' \rightarrow (A \rightarrow G)$. □

Theorem 6. *Given a component K and two contracts C and C', if $K \models C$ and $C \preceq C'$ then $K \models C'$.*

Proof. This follows directly from Definition 25 and Proposition 3. □

Proposition 15. *PSL contract $C = (A, G)$ dominates $C_1 = (A_1, G_1)$ and $C_2 = (A_2, G_2)$ if*

(a) for any I_1 and I_2 such that $\models I_1 \wedge A_1 \rightarrow G_1$ and $\models I_2 \wedge A_2 \rightarrow G_2$ then $\models (I_1 \wedge I_2) \wedge A \rightarrow G$

(b) for any E such that $\models E \rightarrow A$:

(a) for any I_1 such that $\models I_1 \wedge A_1 \rightarrow G_1$ then $\models (E \wedge I_1) \rightarrow A_2$, and
(b) for any I_2 such that $\models I_2 \wedge A_2 \rightarrow G_2$ then $\models (E \wedge I_2) \rightarrow A_1$

Proof. This follows directly from Definition 9, Proposition 10 and Theorem 4. □

Proposition 16. *Given two dominatible PSL contracts $C_1 = (A_1, G_1)$ and $C_2 = (A_2, G_2)$ which have normal forms $\underline{C_1} = (A_1, \underline{G_1})$ and $\underline{C_2} = (A_2, \underline{G_2})$,*

$$C_1 \boxtimes C_2 = (\widetilde{A}, \widetilde{G})$$

where

$$\begin{aligned} \widetilde{A} &= (\underline{G_2} \rightarrow A_1) \wedge (\underline{G_1} \rightarrow A_2) \\ &= ((A_2 \rightarrow G_2) \rightarrow A_1) \wedge ((A_1 \rightarrow G_1) \rightarrow A_2 \end{aligned}$$

and

$$\begin{aligned} \widetilde{G} &= \underline{G_1} \wedge \underline{G_2} \\ &= (A_1 \rightarrow G_1) \wedge (A_2 \rightarrow G_2) \end{aligned}$$

Proof. This follows directly from Definition 11, Theorem 4 and Definition 23. □

Corollary 4. *Given PSL contracts $C = (A, G)$, $C_1 = (A_1, G_1)$ and $C_2 = (A_2, G_2)$. Then $C_1 \boxtimes C_2 \preceq C$ iff $\models A \to \widetilde{A}$ and $\models \widetilde{G} \to \underline{G}$.*

Proof. This follows from Corollary 3 and Proposition 16.

□

Theorem 7. [**Correctness of Algorithm 5**] COMPONENT-VERIF$(K, C) =$ true *if and only if* $K \models C$.

Proof. We prove it by induction on the structure of K.

If K is atomic, then the result of COMPONENT-VERIF(K, C) is the same result of calling the function ATOMIC-COMPONENT-VERIF(K, C) from Algorithm 4, which is $\models \mathsf{implspec}[\![K]\!] \wedge A \to G$, which according to Proposition 10 and Definition 25, is the same as $K \models C$.

If K is composite, then the result of COMPONENT-VERIF(K, C) is the result of calling the function BINARY-COMPONENT-VERIF(K, C) from Algorithm 6. By induction hypothesis, COMPONENT-VERIF$(K_1, C_1) =$ true if and only if $K_1 \models C_1$ and COMPONENT-VERIF$(K_2, C_2) =$ true if and only if $K_2 \models C_2$, in other words, v_1 and v_2 in lines 3 and 4 of Algorithm 6 are true iff $K_i \models C_i$ for $i \in \{1, 2\}$. But this is to say that $\mathsf{implspec}[\![K_1]\!] \in \mathsf{impl}[\![C_1]\!]$ and $\mathsf{implspec}[\![K_2]\!] \in \mathsf{impl}[\![C_2]\!]$ by Definition 25. In other words, K_1 and K_2 are valid implementations of $\hat{C}_1$ and $\hat{C}_2$ respectively. By Theorem 2 and Definition 10 we know that $\hat{C}_1 \boxtimes \hat{C}_2$ dominates both $\hat{C}_1$ and $\hat{C}_2$ which means (Definition 9) that the composition of any implementations of $\hat{C}_1$ and $\hat{C}_2$ is a valid implementation of $\hat{C}_1 \boxtimes \hat{C}_2$, *i.e.*, if $I_1 \in \mathsf{impl}[\![\hat{C}_1]\!]$ and $I_2 \in \mathsf{impl}[\![\hat{C}_2]\!]$ then $I_1 \otimes^{\mathsf{psl}} I_2 \in \mathsf{impl}[\![\hat{C}_1 \boxtimes \hat{C}_2]\!]$. But we have established that $\mathsf{implspec}[\![K_1]\!] \in \mathsf{impl}[\![C_1]\!]$ and $\mathsf{implspec}[\![K_2]\!] \in \mathsf{impl}[\![C_2]\!]$ so $\mathsf{implspec}[\![K_1]\!] \otimes^{\mathsf{psl}} \mathsf{implspec}[\![K_2]\!] \in \mathsf{impl}[\![\hat{C}_1 \boxtimes \hat{C}_2]\!]$, and by Definition 24, $\mathsf{implspec}[\![K_1 \parallel K_2]\!] = \mathsf{implspec}[\![K_1]\!] \otimes^{\mathsf{psl}} \mathsf{implspec}[\![K_2]\!]$. Hence $K = \mathsf{implspec}[\![K_1 \parallel K_2]\!] \in \mathsf{impl}[\![\hat{C}_1 \boxtimes \hat{C}_2]\!]$, which is to say that $K \models \hat{C}_1 \boxtimes \hat{C}_2$. Line 4 of Algorithm 6 establishes whether $\hat{C}_1 \boxtimes \hat{C}_2 \preceq C$. Hence, by Theorem 6 we get $K \models C$. □

Proposition 17. ***[Contract composition is commutative and associative and preserves strong contract equivalence]*** *For any contracts C, C_1, C_2, C_3 in a simplified specification theory:*

(i) $C_1 \boxtimes C_2 \equiv C_2 \boxtimes C_1$

(ii) $(C_1 \boxtimes C_2) \boxtimes C_3 \simeq C_1 \boxtimes (C_2 \boxtimes C_3)$

(iii) if $C_1 \equiv C_2$ and C_1 and C_2 are in normal form, then $C_1 \boxtimes C \equiv C_2 \boxtimes C$

Proof.

(i) By commutativity of specifications:

$$\begin{aligned} C_1 \boxtimes C_2 &\equiv ((A_1/\underline{G_2}) \wedge (A_2/\underline{G_1}), \underline{G_1} \otimes \underline{G_2}) \\ &\equiv ((A_2/\underline{G_1}) \wedge (A_1/\underline{G_2}), \underline{G_2} \otimes \underline{G_1}) \\ &\equiv C_2 \boxtimes C_1 \end{aligned}$$

(ii) Let us define $C \stackrel{def}{=} C_1 \boxtimes C_2 = (A, G)$, $C' \stackrel{def}{=} C_2 \boxtimes C_3 = (A', G')$, $C_L \stackrel{def}{=} C \boxtimes C_3 = (A_L, G_L)$ and $C_R \stackrel{def}{=} C_1 \boxtimes C' = (A_R, G_R)$. Then, we need to prove that $C_L \simeq C_R$. We prove $C_L \preceq C_R$, and $C_R \preceq C_L$ follows by a symmetric argument. To prove $C_L \preceq C_R$, it is enough, by Theorem 1, to prove that

(a) $A_R \leqslant A_L$ and (b) $G_L \leqslant_{A_R} G_R$. First, let us spell out the contract compositions above, according to Definition 11:

$$\begin{aligned} A &= (A_1/\underline{G_2}) \wedge (A_2/\underline{G_1}) \\ G &= \underline{G_1} \otimes \underline{G_2} \\ A' &= (A_2/\underline{G_3}) \wedge (A_3/\underline{G_2}) \\ G' &= \underline{G_2} \otimes \underline{G_3} \\ A_L &= (A/\underline{G_3}) \wedge (A_3/\underline{G}) \\ G_L &= \underline{G} \otimes \underline{G_3} \\ A_R &= (A_1/\underline{G'}) \wedge (A'/\underline{G_1}) \\ G_R &= \underline{G_1} \otimes \underline{G'} \end{aligned}$$

Note that $\underline{G} = G/A$ and $\underline{G'} = G'/A'$, in accordance to Definition 15.

(a) Let us expand A_L, first by considering $A/\underline{G_3}$, using Proposition 7 and Proposition 1.

$$\begin{aligned} A/\underline{G_3} &= ((A_1/\underline{G_2}) \wedge (A_2/\underline{G_1}))/\underline{G_3} \\ &= ((A_1/\underline{G_2})/\underline{G_3}) \wedge ((A_2/\underline{G_1})/\underline{G_3}) \\ &= (A_1/(\underline{G_2} \wedge \underline{G_3})) \wedge (A_2/(\underline{G_1} \wedge \underline{G_3})) \end{aligned}$$

Now, since $A_3/\underline{G} = A_3/(\underline{G_1} \wedge \underline{G_2})$ we have

$$\begin{aligned} A_L &= (A/\underline{G_3}) \wedge (A_3/\underline{G}) \\ &= (A_1/(\underline{G_2} \wedge \underline{G_3})) \wedge (A_2/(\underline{G_1} \wedge \underline{G_3})) \wedge A_3/(\underline{G_1} \wedge \underline{G_2}) \end{aligned}$$

Now we expand A_R, first by considering $A'/\underline{G_1}$:

$$\begin{aligned} A'/\underline{G_1} &= ((A_2/\underline{G_3}) \wedge (A_3/\underline{G_2}))/\underline{G_1} \\ &= ((A_2/\underline{G_3})/\underline{G_1}) \wedge ((A_3/\underline{G_2})/\underline{G_1}) \\ &= (A_2/(\underline{G_3} \wedge \underline{G_1})) \wedge (A_3/(\underline{G_2} \wedge \underline{G_1})) \\ &= (A_2/(\underline{G_1} \wedge \underline{G_3})) \wedge (A_3/(\underline{G_1} \wedge \underline{G_2})) \end{aligned}$$

Now, since $A_1/\underline{G'} = A_1/(\underline{G_2} \wedge \underline{G_3})$ we have that

$$\begin{aligned} A_R &= (A_1/\underline{G'}) \wedge (A'/\underline{G_1}) \\ &= A_1/(\underline{G_2} \wedge \underline{G_3}) \wedge ((A_2/(\underline{G_1} \wedge \underline{G_3})) \wedge (A_3/(\underline{G_1} \wedge \underline{G_2}))) \\ &= A_L \end{aligned}$$

Hence $A_R \leqslant A_L$.

(b) To prove $G_L \leqslant_{A_R} G_R$ it is enough to prove that $G_L \otimes A_R \leqslant G_R$ by Lemma 1. We will prove this by showing that $G_L \leqslant G_R/A_R$. Since $\underline{G} = G/A$ we have

$$\begin{aligned} G_L &= \underline{G} \otimes \underline{G_3} \\ &= (G/A) \otimes \underline{G_3} \\ &= ((\underline{G_1} \otimes \underline{G_2})/A) \otimes \underline{G_3} \\ &\leqslant ((\underline{G_1} \otimes \underline{G_2}) \otimes \underline{G_3})/A \end{aligned}$$

Since $\underline{G'} = G'/A'$ we have

$$\begin{aligned} G_R/A_R &= (\underline{G_1} \otimes \underline{G'})/A_R \\ &= (\underline{G_1} \otimes (G'/A'))/A_R \\ &= (\underline{G_1}/A_R) \otimes ((G'/A')/A_R) \\ &= (\underline{G_1}/A_R) \otimes (G'/(A' \wedge A_R)) \\ &\geqslant (\underline{G_1}/A_R) \otimes (G'/A_R) \\ &= (\underline{G_1} \otimes G')/A_R \\ &= (\underline{G_1} \otimes (\underline{G_2} \otimes \underline{G_3}))/A_R \end{aligned}$$

(iii) Since $C \simeq C$ by reflexivity, and since $C_1 \equiv C_2$ implies $C_1 \simeq C_2$ by Proposition 5 then $C_1 \boxtimes C \simeq C_2 \boxtimes C$ by Corollary 1. Hence, $C_1 \boxtimes C \equiv C_2 \boxtimes C$ because the contracts are in normal form (Proposition 5).

□

Proposition 18. *If $n > 1$ then $\boxtimes_{i=1}^{n} C_i \equiv C_1 \boxtimes (\boxtimes_{i=2}^{n} C_i)$*

Proof. We prove this by induction on n:

Case 1. $n = 2$: Then $\boxtimes_{i=1}^{n} C_i \equiv (\boxtimes_{i=1}^{n-1} C_i) \boxtimes C_n \equiv (\boxtimes_{i=1}^{2-1} C_i) \boxtimes C_2 \equiv C_1 \boxtimes C_2 \equiv C_1 \boxtimes (\boxtimes_{i=2}^{2} C_i)$ by Definition 28.

Case 2. $n > 2$: Then, assume that the statement holds for all $m < n$ (induction hypothesis)

$$\begin{aligned} \boxtimes_{i=1}^{n} C_i &\equiv (\boxtimes_{i=1}^{n-1} C_i) \boxtimes C_n && \text{by Definition 28} \\ &\equiv (C_1 \boxtimes (\boxtimes_{i=2}^{n-1} C_i)) \boxtimes C_n && \text{by induction hypothesis and Proposition 17(iii)} \\ &\equiv C_1 \boxtimes ((\boxtimes_{i=2}^{n-1} C_i) \boxtimes C_n) && \text{by associativity of } \boxtimes \\ &\equiv C_1 \boxtimes ((\boxtimes_{j=1}^{n-2} C_{j+1}) \boxtimes C_n) && \text{by changing of variables } j \stackrel{def}{=} i-1 \\ &\equiv C_1 \boxtimes ((\boxtimes_{j=1}^{n-2} C'_j) \boxtimes C'_{n-1}) && \text{by defining } C'_j \stackrel{def}{=} C_{j+1} \\ &\equiv C_1 \boxtimes (\boxtimes_{j=1}^{n-1} C'_j) && \text{by Definition 28} \\ &\equiv C_1 \boxtimes (\boxtimes_{i=2}^{n-1} C_i) \end{aligned}$$

□

Proposition 19. *Let $I = \{1, ..., n\}$ and $\{C_i\}_{i \in I}$ a family of contracts. Then $\boxtimes_{i \in I} C_i = (\widetilde{A}, \widetilde{G})$ where*

$$\widetilde{A} = \bigwedge_{i \in I} (A_i / (\bigwedge_{j \in I \setminus \{i\}} G_i))$$

and

$$\widetilde{G} = \bigwedge_{i \in I} G_i$$

Proof. a □

C.5 Quotienting

Theorem 8. *Given a pair of contracts $C_0 = (A_0, G_0)$ and $C_1 = (A_1, G_1)$, if C_0 and C_1 are in normal form, then $C_1 \boxtimes (C_0/C_1) \preceq C_0$.*

Proof. Let us call $C_X = C_0/C_1$ and $C' = C_1 \boxtimes C_X$. By definition of contract quotient (Definition 29) we know that $C_X = (\ddot{A_X}, \ddot{G_X})$ where $\ddot{A_X} = G_1 \otimes A_0$ and $\ddot{G_X} = (G_0/G_1) \wedge (A_1/A_0)$. Hence, by definition of contract composition (Definition 10) we have that $C' = (\widetilde{A'}, \widetilde{G'})$ where $\widetilde{A'} = (A_1/\ddot{G_X}) \wedge (\ddot{A_X}/\underline{G_1})$ and $\widetilde{G'} = \underline{G_1} \otimes \ddot{G_X}$. In order to prove that $C' \preceq C$, by using Theorem 1 it is enough to prove that (1) $A_0 \leqslant \widetilde{A'}$ and (2) $\widetilde{G'} \leqslant_{A_0} G_0$.

1. To show that $A_0 \leqslant \widetilde{A'}$ we need to show that (a) $A_0 \leqslant A_1/\ddot{G_X}$ and that (b) $A_0 \leqslant \ddot{A_X}/\underline{G_1}$. The result then follows by Definition 1(A7).

   (a) By Definition 1(A3) we know that $A_0 \otimes (A_1/A_0) \leqslant A_1$, and by commutativity of $\otimes$, we have that $(A_1/A_0) \otimes A_0 \leqslant A_1$. By (A6) we also know that $(G_0/G_1) \wedge (A_1/A_0) \leqslant (A_1/A_0)$, which by (A1) entails $((G_0/G_1) \wedge (A_1/A_0)) \otimes A_0 \leqslant (A_1/A_0) \otimes A_0$ and by transitivity of $\leqslant$ we obtain $((G_0/G_1) \wedge (A_1/A_0)) \otimes A_0 \leqslant A_1$. Since $\ddot{G_X} = (G_0/G_1) \wedge (A_1/A_0)$, we have proven that $\ddot{G_X} \otimes A_0 \leqslant A_1$ which, by (A4) implies that $A_0 \leqslant A_1/\ddot{G_X}$.

   (b) By reflexivity we know that $G_1 \otimes A_0 \leqslant G_1 \otimes A_0$ and so by Definition 1(A3) we know that $A_0 \leqslant (G_1 \otimes A_0)/G_1$. But we defined $\ddot{A_X} = G_1 \otimes A_0$ so we have proven that $A_0 \leqslant \ddot{A_X}/G_1$ as required.

2. Since C_0 is in normal form, by Definition 8, proving $\widetilde{G'} \leqslant_{A_0} G_0$ is the same as proving $\widetilde{G'} \leqslant G_0$. We show this as follows. By Definition 1(A3) we know that $G_1 \otimes (G_0/G_1) \leqslant G_0$, and by (A6) we also know that $(G_0/G_1) \wedge (A_1/A) \leqslant G_0/G_1$. This implies by (A1) that $G_1 \otimes ((G_0/G_1) \wedge (A_1/A)) \leqslant G_1 \otimes (G_0/G_1)$ which by transitivity entails that $G_1 \otimes ((G_0/G_1) \wedge (A_1/A)) \leqslant G_0$, but $\ddot{G_X} = (G_0/G_1) \wedge (A_1/A_0)$ so we have shown that $G_1 \otimes \ddot{G_X} \leqslant G_0$, and $\widetilde{G'} = \underline{G_1} \otimes \ddot{G_X}$, therefore $\widetilde{G'} \leqslant G_0$ as required.

□