Parametrised Compositional Verification with Multiple Process and Data Types

Antti Siirtola, Keijo Heljanko
Department of Information and Computer Science, Aalto University, P.O.Box 15400, FI-00076 Aalto, Finland
firstname.lastname(at)aalto.fi

***Abstract*—We present an LTS-based (Labelled Transition System) CSP-like (Communicating Sequential Processes) formalism for expressing parametrised systems. The parameters are process types, which determine the number of replicated components, and data types, which enable components with a parametrised state space. We prove that the formalism is compositional and show how to combine two existing results for parametrised verification in order to check trace refinement between parametrised processes. The combined approach gives upper bounds, i.e., cut-offs, for types such that a parametrised verification task collapses into finitely many checks solvable by using existing finite state refinement checking tools. We have implemented the approach and applied it to prove mutual exclusion properties of network protocols and systems with shared resources. To the best our knowledge, our technique is the only one that combines compositionality and completeness with support for multiple parametric process and data types.**

I. Introduction

Concurrent and object-oriented software systems have many natural parameters which can take infinitely many values. Such parameters are the number of replicated parts, like processes and objects, and the size of data types, like message domains and address spaces. Consequently, the question on the correctness of a software system is naturally expressed as the *parametrised verification problem*: given a parametrised system implementation and specification, determine whether the implementation is correct with respect to the specification for all parameter values.

In practice, we can automatically verify only the smallest instances of a parametrised system in order to detect bugs in the implementation. Sometimes, we can also establish upper bounds, i.e., *cut-offs*, for the parameters such that the system implementation is correct for all parameter values if and only if it is correct for all values up to the cut-offs. In other words, if there is a bug in an implementation instance with a parameter value greater than the cut-off, then there is an analogous bug in an implementation instance where the values of the parameters are within the cut-offs. Although such cut-offs always exist, determining them algorithmically is undecidable in the most general case [1].

Another practical challenge is that some subsystems (e.g., external software packages and subsystems concurrently under construction) can be available only in interface specification form. Hence, there is a need to perform parametrised verification in a *compositional* way, where we first substitute the subsystem specification for each subsystem implementation that is known or proved to be correct and then analyse the correctness of the whole system. This is possible if the correctness relation is a *precongruence*, a reflexive and transitive relation which is preserved under the application of composition operators.

Contribution: As the first contribution, we present a formalism which enables expressing parametrised systems and specifications and allows for *compositional* analysis and design. The formalism is based on an LTS-based (Labelled Transition System) CSP-like [2], [3] (Communicating Sequential Processes) process calculus with hiding and parallel composition operators, which we parametrise with types and variables of two kinds. Process types enable parametrising the structure of a system by controlling the number of replicated concurrent parts. Data types allow for parametrising the state space of components by controlling the number of similar transitions acting on different data. Process and data variables refer to individual components and data values, respectively, and their values can be tested for (in)equality. Hence, the formalism is like the one in [4] but significantly extended with data parametrisation features. It allows for expressing interesting and practically important classes of systems and specifications, like the mutual exclusion properties of network protocols and systems with shared resources, but increasing its expressiveness further quickly leads to undecidability [5].

As the main contribution, we show how to compute cut-offs for trace refinement checking between parametrised processes. This is done by combining two existing results on parametrised verification: the *precongruence reduction (PR)* method of Siirtola and Kortelainen [6], [7], [5], [4] and the *data independence (DI)* results of Lazić and Novak [8], [9]. It is not obvious that the combination of the results leads to something useful since the PR technique applies to systems with process types but does not allow for components with a parametrised state space, whereas the DI results apply to systems with data types but do not allow for systems with a parametric number of replicated components.

Here, we solve the problem by bounding the process types with the PR technique first. Basically, we should fix the values of data types before we can apply the PR technique, which means that we ought to apply the technique infinitely many times in order to determine cut-offs for process types. However, since the data types can affect the structure of the system only finitely many ways, it turns out that we

can compute rough structural cut-offs for process types as before without paying attention to data types. After that, we can basically substitute the parallel composition of finitely many parametrised processes for a replicated parametrised parallel composition and apply the DI results to bound the size of data types. For each type T, the cut-off size can be computed easily based on the structure of the parametrised system implementation and specification and the number of variables of the type T. After that, the cut-offs can be further improved by analysing the instances up to the cut-offs at the operational level. Finally, the verification can be completed by solving the remaining finitely many finite state refinement checking tasks by using existing tools.

As a running example, we consider a host configuration protocol (HCP) with an arbitrary number of hosts (process type) and an arbitrary number of network addresses (data type). Since the protocol involves types of both kinds, it cannot be handled by either technique alone. However, it can be treated with our combined result, which implies that the result presented here is more powerful than either of the approaches alone. We have implemented the technique and, in addition to HCP, applied it to establish mutual exclusion properties for systems with shared resources. Since our formalism is compositional, it means that the specifications of these systems can be used in place of the implementations in further verification efforts.

Except for some small differences in formalism and the definition of cut-offs, our DI results are semantically similar to those of Lazić and Novak [8], [9]. Additionally, our DI proof technique is simpler than in [8], [9]: we use only the bijective mapping of transitions and avoid the construction of intermediate structures, like exploration graphs [8] and quotient LTSs [9], altogether. This leads to a more accessible and concise theory of data independence, which is the third contribution of the paper. However, unlike [8], [9], we consider only safety properties, since the PR technique cannot be extended to deadlock preserving semantics directly [5]. On the other hand, extending the DI part of our results to such semantics should be easy, since all the key DI lemmas are formulated at the operational level.

Related Work: The distinctive features of our approach are compositionality, completeness and support for the multiple parameters of two fundamentally different types.

Parametrised verification in a compositional setting is previously considered by Valmari & Tienari [10] and Creese [11]. In both works, induction is exploited to determine an abstract process which can be used in verification in place of the parametrised subsystem. The methods scale to the multi-parametrised case, too, but they involve the discovery of an invariant process which is a task that cannot be automated in general.

Ghilardi and Ranise present a complete parametrised verification technique [12], where the infinite family of finite state systems arising from a parametrised system description is encoded as a single infinite state well-structured transition system (WSTS) [13]. The technique allows for two levels of parametrisation, the number and the state space of processes, but since the formalism is not compositional, it does not allow for an arbitrary number of nested parametrised substructures like ours. The completeness of the approach is based on the fact that there is an order on the set of states compatible with the transition relation [13]. In our case, there is no such a criterion nor an (obvious) way to define a compatible order among the states of all the instances. That is because our processes communicate through alphabet-based synchronisation, where increasing the number of replicated components may block some transitions and hence break compatibility.

Other methods that enable multi-parametrised verification are by Emerson & Kahlon [14], Yang & Li [15] and Hanna et al. [16]. Their approaches apply to systems with guarded broadcasts [14], shared actions [16] or rendezvous communication [15]. The methods are cut-off-based and allow for an arbitrary number of parameters which specify the number of replicated processes. However, none of them supports compositional analysis nor parametrised data types.

Other complete parametrised verification methods, in turn, are based on either cut-offs [17], [18], [19], [20], [21] or WSTSs [22], which covers counter abstraction and Petri nets [23]. WSTSs are already discussed earlier and the other cut-off results are only for closed systems with a single parameter determining the number of replicated processes.

Moreover, some of the results can be obtained in our approach, too. At least the systems with conjunctive guards [14] and homogeneous processes communicating through prioritised queue policy [21] can be modelled in our formalism [5]. On the one hand, most cut-off results including [14], [21] allow for the analysis of liveness, too, whereas our result as well as WSTS-based approaches are restricted to safety properties. On the other hand, parametrised systems where replicated processes communicate through rendezvous events are difficult to model in our formalism, whereas they can often be modelled as WSTSs.

Outline: In Section II, we introduce our process model. After that, we parametrise it while preserving compositionality. In Section IV, we present the cut-off results and finally, the paper concludes with discussion on future work. For the sake of readability and the lack of space, proofs are found in the online appendix [24].

II. Processes

In this section, we define a fairly standard notation used throughout the paper and a CSP- and LTS-based process model with parallel composition and hiding operators. The main difference with the standard LTSs is that in our model, states and events have an explicit data part which makes adding parametrisation convenient.

Notation: For any set A, A^* denotes the set of all finite tuples over A, especially the empty tuple $()$ is in A^*. Whenever $\mathbf{a} := (a_1, \ldots, a_n)$ and $\mathbf{b} := (b_1, \ldots, b_k)$ are tuples over A, then $|\mathbf{a}|$ denotes the *length* n of $\mathbf{a}$, and $\mathbf{ab}$ the *concatenation* $(a_1, \ldots, a_n, b_1, \ldots, b_k)$ of $\mathbf{a}$ and $\mathbf{b}$. For any function $f : A \mapsto B$, $\mathrm{dom}(f)$ is the *domain* A of f, and $\mathrm{im}(f)$ the *image* $\{f(a) \mid a \in A\}$ of f. The tuple $\mathbf{a}$ can be regarded as a function $f : \{1, \ldots, n\} \mapsto A$ such that $f(i) = a_i$ for all $i \in \{1, \ldots, n\}$, so we can define $\mathrm{im}(\mathbf{a})$ as the set $\{a_i \mid i = 1, \ldots, n\}$. For any set C, $f|_C$ denotes the *restriction* of f to $A \cap C$, i.e., a function $f' : A \cap C \mapsto B$ such that $f'(a) = f(a)$ for all $a \in A \cap C$. For any function g such that $\mathrm{im}(f) \subseteq \mathrm{dom}(g)$, $g \circ f$ is the *composition* of g and f, i.e., a function $h : \mathrm{dom}(f) \mapsto \mathrm{im}(g)$ such that $h(a) = g(f(a))$ for all $a \in A$.

Processes: A process is basically an LTS [3] where states have a control and data part and events have a channel and data part. Let $\mathbb{S}$, $\mathbb{C}$ and $\mathbb{V}$ be mutually disjoint countably infinite sets of *control states*, *channels* and *constants*, respectively, such that $\mathbb{S}$ contains the natural numbers and is closed under the Cartesian product. We assume that there is a single *invisible* channel $\tau \in \mathbb{C}$ which represents the internal activity of a process. The other channels are called *visible* and they are used for communication among processes and environment. A structure $s\,\mathbf{d}$, where $s \in \mathbb{S}$ and $\mathbf{d} \in \mathbb{V}^*$, is a *state* and a structure $c\,\mathbf{e}$, where $c \in \mathbb{C}$ and $\mathbf{e} \in \mathbb{V}^*$, is an *event*. If $c \neq \tau$, the event is *visible*, otherwise it is *invisible*.

Definition 1 (Process)**.** A *process* P is a pair $(\hat{s}\,\hat{\mathbf{d}}, R)$, where $\hat{s}\,\hat{\mathbf{d}}$ is a state and R is a set of triples $(s\,\mathbf{d}, c\,\mathbf{e}, s'\mathbf{d}')$ such that $s\,\mathbf{d}, s'\mathbf{d}'$ are states and $c\,\mathbf{e}$ is an event.

The first component of P is called the *initial state*, denoted by $\mathrm{init}(P)$, and the second one is the set of *transitions*, denoted by $\mathrm{tran}(P)$. The set of all the *visible* events occurring in P, the *alphabet* of P, is denoted by $\mathrm{alph}(P)$. P is *(in)finite* if and only if $\mathrm{tran}(P)$ is. P is *(strongly) deterministic* if (i) it does not involve invisible events and (ii) whenever $(s\,\mathbf{d}, c_1\,\mathbf{e}_1, s_1\,\mathbf{d}_1)$ and $(s\,\mathbf{d}, c_2\,\mathbf{e}_2, s_2\,\mathbf{d}_2)$ are transitions of P such that $s_1\,\mathbf{d}_1 \neq s_2\,\mathbf{d}_2$, then $c_1\,\mathbf{e}_1 \neq c_2\,\mathbf{e}_2$.

Operators: In our formalism, both a system implementation and specification are modelled as processes P_{impl} and P_{spec}, respectively. Since the implementation often consists of several components, P_{impl} is typically a parallel composition of smaller processes, and before P_{impl} is compared against P_{spec}, the channels irrelevant to P_{spec} are hidden.

Definition 2 (Parallel Composition)**.** Let P_i be the process $(\hat{s}_i\,\hat{\mathbf{d}}_i, R_i)$ for both $i \in \{1, 2\}$. The *parallel composition (of P_1 and P_2)*, denoted by $(P_1 \parallel P_2)$, is a pair $((\hat{s}_1, \hat{s}_2, |\hat{\mathbf{d}}_1|)\,\hat{\mathbf{d}}_1\hat{\mathbf{d}}_2, R_\parallel)$, where $R_\parallel$ is the set of all triples $((s_1, s_2, |\mathbf{d}_1|)\,\mathbf{d}_1\mathbf{d}_2, c\,\mathbf{e}, (s_1', s_2', |\mathbf{d}_1'|)\,\mathbf{d}_1'\mathbf{d}_2')$ such that (i) $c \neq \tau$ and $(s_i\,\mathbf{d}_i, c\,\mathbf{e}, s_i'\,\mathbf{d}_i') \in R_i$ for both $i \in \{1, 2\}$, or (ii) $(s_i\,\mathbf{d}_i, c\,\mathbf{e}, s_i'\,\mathbf{d}_i') \in R_i$, $c\,\mathbf{e} \notin \mathrm{alph}(P_j)$, $s_j\,\mathbf{d}_j$ is a state of P_j and $s_j\,\mathbf{d}_j = s_j'\,\mathbf{d}_j'$ for different elements $i, j \in \{1, 2\}$. Obviously, $(P_1 \parallel P_2)$ is a process, where P_1 and P_2 execute a visible event jointly if and only if both agree on its execution, whereas the visible events only in the alphabet of one process and the invisible events are executed individually. This is essentially the parallel composition operator of CSP [2], [3] where the synchronisation alphabet is $\mathrm{alph}(P_1) \cap \mathrm{alph}(P_2)$. Note that the third component in the control states of $(P_1 \parallel P_2)$ tells which part of the data vector comes from P_1. Without the third component, the composition of states $s_1(a)$ and $s_2()$ and states $s_1()$ and $s_2(a)$ would be the same, which is not what we want.

Definition 3 (Hiding)**.** Let P be the process $(\hat{s}\,\hat{\mathbf{d}}, R)$ and E a set of visible channels. The process P *after hiding* E, denoted by $(P \setminus E)$, is a pair $(\hat{s}\,\hat{\mathbf{d}}, R_\setminus)$, where $R_\setminus$ is the set of (i) all triples $(s\,\mathbf{d}, c\,\mathbf{e}, s'\mathbf{d}') \in R$ such that $c \notin E$ and (ii) all triples $(s\,\mathbf{d}, \tau\mathbf{e}, s'\mathbf{d}')$ such that $(s\,\mathbf{d}, c\,\mathbf{e}, s'\mathbf{d}') \in R$ for some $c \in E$.

In other words, $(P \setminus E)$ is a process obtained from P by substituting τ for the channels c in E.

Semantics: For verification purposes, a process is interpreted as a set of traces, which enables us to consider safety properties. A finite alternating sequence $(s_0\,\mathbf{d}_0, c_1\,\mathbf{e}_1, s_1\,\mathbf{d}_1, \ldots, c_n\,\mathbf{e}_n, s_n\,\mathbf{d}_n)$ of states and events of P is a *path of P (from $s_0\,\mathbf{d}_0$) (to $s_n\,\mathbf{d}_n$)* if $(s_{i-1}\,\mathbf{d}_{i-1}, c_i\,\mathbf{e}_i, s_i\,\mathbf{d}_i)$ is a transition of P for every $i \in \{1, \ldots, n\}$. A path from the initial state is called an *execution (of P)*. A finite sequence of visible events is a *trace (of P)*, if there is an execution of P such that the sequence is obtained from the execution by erasing all the states and the invisible events. The set of all the traces of P is denoted by $\mathrm{tr}(P)$. A process P_1 is a *trace refinement* of a process P_2, denoted by $P_1 \preceq_{\mathrm{tr}} P_2$, if and only if $\mathrm{alph}(P_1) = \mathrm{alph}(P_2)$ and $\mathrm{tr}(P_1) \subseteq \mathrm{tr}(P_2)$ [2]. The processes P_1 and P_2 are *trace equivalent*, denoted by $P_1 \equiv_{\mathrm{tr}} P_2$, if and only if $P_1 \preceq_{\mathrm{tr}} P_2$ and $P_2 \preceq_{\mathrm{tr}} P_1$. P_{impl} is considered correct with respect to P_{spec} if and only if $P_{impl} \preceq_{\mathrm{tr}} P_{spec}$.

Calculus: The operators and the trace relations have many useful properties from the viewpoint of synthesis and analysis. First, the parallel composition is commutative and associative with respect to $\equiv_{\mathrm{tr}}$ and a single state process $P_{id} := (s(), \emptyset)$ without transitions is an identity element of the parallel composition. That is why for every finite set $I = \{i_1, \ldots, i_n\}$ and any processes $P_{i_1}, \ldots, P_{i_n}$, we can define the I-indexed version of $\parallel$ as follows: $(\|_{i \in I}\, P_i) := P_{i_1} \parallel (\|_{i \in \{i_2, \ldots, i_n\}}\, P_i)$, when $n > 0$, and $(\|_{i \in I}\, P_i) := P_{id}$, when $n = 0$. Secondly, $\preceq_{\mathrm{tr}}$ is a preorder (i.e., a reflexive and transitive relation) and $\equiv_{\mathrm{tr}}$ an equivalence on the set of processes. Moreover, $\preceq_{\mathrm{tr}}$ is *compositional*, i.e., preserved under the application of the operators: if $P_1 \preceq_{\mathrm{tr}} P_2$, then $P_1 \parallel P \preceq_{\mathrm{tr}} P_2 \parallel P$, $P \parallel P_1 \preceq_{\mathrm{tr}} P \parallel P_2$ and $P_1 \setminus E \preceq_{\mathrm{tr}} P_2 \setminus E$ for all processes P_1, P_2, P and sets E of visible channels. Hence, $\preceq_{\mathrm{tr}}$ is a compositional preorder, a *precongruence*, on the set of processes.

III. Parametrised Processes

In this section, we equip our process calculus with parameters. Our parametrised formalism is based on the one which is presented in [4] and which allows for parametrising the structure of a system. Here, we extend that formalism with data parameters which allow for components with a parametrised state space, too.

Running Example: As a running example, we consider a host configuration protocol (HCP), where each host repeatedly picks a network address until it finds one that is not used by other hosts. This is done by broadcasting address queries and replies to other hosts in the style of ARP (Address Resolution Protocol). Our goal is to formally model the protocol with an arbitrary number of hosts and an arbitrarily large address space and prove that in our construction, each address is possessed by at most one host.

Parameters: In order to model HCP, we parametrise processes and operators with four kinds of variables: *(process and data) types* and *(process and data) variables*. A data type denotes a finite non-empty set of data values and a process type represents a finite non-empty set of the identifiers of replicated components of a certain kind whereas a process variable refers to the identifier of an individual component and a data variable to a data value. A (concurrent) parametrised system is composed of sequential parametrised processes each of which represents the system from the viewpoint of finitely many replicated components.

Formally, we assume that for each type T there is a countably infinite set $I_T \subseteq \mathbb{V}$ of constants such that I_U and I_V are disjoint whenever U and V are different types, and for each process and data variable x, there is respectively a unique process or data type T_x. The possible values of a type T are the finite non-empty subsets of I_T and the possible values of a variable x are the elements of I_{T_x}. We assume that the sets of process types, data types, process variables and data variables, denoted by $\mathbb{T}_P$, $\mathbb{T}_D$, $\mathbb{X}_P$ and $\mathbb{X}_D$, respectively, are disjoint and countably infinite. We write $\mathbb{T}$ and $\mathbb{X}$ short for $\mathbb{T}_P \cup \mathbb{T}_D$ and $\mathbb{X}_P \cup \mathbb{X}_D$, respectively.

In HCP, there are two kinds of replicated objects: hosts and addresses. Hence, we pick a process type T_H to represent the set of the identifiers of hosts and a data type T_A to denote the set of available addresses.

Guards: In our parametrised formalism, (in)equality tests between variables are represented as guards.

Definition 4 (Guard)**.** *Guards* $\mathcal{C}$ are given by the grammar

$$\mathcal{C} ::= \top \mid x \mathrel{\widehat{=}} y \mid (\widehat{\neg}\, \mathcal{C}) \mid (\mathcal{C} \mathrel{\widehat{\wedge}} \mathcal{C}),$$

where x and y range over variables.

The *parameters* of a guard $\mathcal{C}$ are the variables occurring in $\mathcal{C}$. The set of all the parameters of $\mathcal{C}$ is denoted by $\mathrm{par}(\mathcal{C})$ and the set $\mathrm{par}(\mathcal{C}) \cup \{T_x \mid x \in \mathrm{par}(\mathcal{C})\}$ of all the parameters of $\mathcal{C}$ plus the types of the parameters is denoted by $\overline{\mathrm{par}}(\mathcal{C})$.

Intuitively, $\top$ denotes a guard which is always true and symbols with a hat on top are interpreted as connectives without one; the hat is included just to mark operators which act on parametrised structures. Formally, a guard is instantiated by using a function called a valuation which assigns values to parameters.

Definition 5 (Valuation)**.** A *valuation* is a function ϕ whose domain is a finite set of types and variables such that (i) for each type $T \in \mathrm{dom}(\phi)$, $\phi(T)$ is a finite non-empty subset of I_T and (ii) for every variable $x \in \mathrm{dom}(\phi)$, $T_x \in \mathrm{dom}(\phi)$ and $\phi(x) \in \phi(T_x)$.

A valuation ϕ is *compatible* with a guard $\mathcal{C}$ if and only if $\mathrm{par}(\mathcal{C}) \subseteq \mathrm{dom}(\phi)$, i.e., $\overline{\mathrm{par}}(\mathcal{C}) \subseteq \mathrm{dom}(\phi)$.

Definition 6 (Instance of Guard)**.** Let $\mathcal{C}$ be a guard and ϕ a compatible valuation. The *(ϕ-)instance of* $\mathcal{C}$, denoted by $[\![\mathcal{C}]\!]_\phi$, is determined inductively as follows:

1) $[\![\top]\!]_\phi$ is true,
2) $[\![x \mathrel{\widehat{=}} y]\!]_\phi$ equals $\phi(x) = \phi(y)$,
3) $[\![\widehat{\neg}\, \mathcal{C}']\!]_\phi$ is $\neg\, [\![\mathcal{C}']\!]_\phi$ and
4) $[\![\mathcal{C}_1 \mathrel{\widehat{\wedge}} \mathcal{C}_2]\!]_\phi = [\![\mathcal{C}_1]\!]_\phi \wedge [\![\mathcal{C}_2]\!]_\phi$.

Parametrised Processes: A structure $s\,\mathbf{x}$, where $s \in \mathbb{S}$ and $\mathbf{x}$ is a tuple of *data* variables, is a *parametrised state*. Respectively, a structure $c\,\mathbf{y}$, where $c \in \mathbb{C}$ and $\mathbf{y}$ is a tuple of *any* variables, is a *parametrised event*. A sequential parametrised process is basically a process where parametrised states and events are substituted for the ordinary ones and transitions are equipped with a guard and a *choice set*, i.e., a set of *data* variables whose values are fixed during the execution of the transition.

Definition 7 (SPP)**.** A sequential parametrised process (SPP) is a pair $\mathcal{S} := (\hat{s}\,\hat{\mathbf{x}}, \Delta)$, where $\hat{s}\,\hat{\mathbf{x}}$ is a parametrised state and Δ is a finite set of five-tuples $(s\,\mathbf{x}, X, \mathcal{C}, c\,\mathbf{y}, s'\mathbf{x}')$ such that $s\,\mathbf{x}$ and $s'\mathbf{x}'$ are parametrised states, $c\,\mathbf{y}$ is a parametrised event, $\mathcal{C}$ is a guard and $X \subseteq \mathrm{im}(\mathbf{y}\mathbf{x}') \cap \mathbb{X}_D$ is a choice set.

The first component of $\mathcal{S}$ is called the *parametrised initial state* and the elements of Δ are *parametrised transitions*. Let x be a variable that occurs in a parametrised transition $(s\,\mathbf{x}, X, \mathcal{C}, c\,\mathbf{y}, s'\mathbf{x}')$. If $x \in \mathrm{im}(\mathbf{x}) \cup X$, then x and all its occurrences are *bound* in the parametrised transition, otherwise they are *free*. A variable is *free* in an SPP $\mathcal{S}$, if it occurs in the parametrised initial state of $\mathcal{S}$ or it is free in some parametrised transition of $\mathcal{S}$. Respectively, a variable is *bound* in $\mathcal{S}$, if it is bound in some parametrised transition of $\mathcal{S}$. Hence, if a variable occurs twice in $\mathcal{S}$, it can be both free and bound. The *parameters* of $\mathcal{S}$ are the *free* variables and the types of *bound* variables occurring in $\mathcal{S}$. Like in the case of guards, $\mathrm{par}(\mathcal{S})$ denotes the set of all the parameters of $\mathcal{S}$, $\overline{\mathrm{par}}(\mathcal{S})$ is the set $\mathrm{par}(\mathcal{S}) \cup \{T_x \mid x \in \mathrm{par}(\mathcal{S}) \cap \mathbb{X}\}$ of all the parameters of $\mathcal{S}$ plus the types of the parameters, and a valuation ϕ is *compatible* with $\mathcal{S}$ if and only if $\mathrm{par}(\mathcal{S}) \subseteq \mathrm{dom}(\phi)$, i.e., $\overline{\mathrm{par}}(\mathcal{S}) \subseteq \mathrm{dom}(\phi)$.

Analogously to predicate logic, the free occurrences of variables are initialised (instantiated) in the beginning and preserve their value throughout the computation whereas the values of the bound occurrences are determined at run time. To put it more formally, let ϕ be a valuation and X a set of variables such that $T_x \in \mathrm{dom}(\phi)$ for every $x \in X$. We write $\mathrm{ext}(\phi, X)$ for the set of all valuations ϕ' with the domain $\mathrm{dom}(\phi) \cup X$ such that ϕ and ϕ' agree on the values of parameters outside X, i.e., $\phi'|_{\mathrm{dom}(\phi)\setminus X} = \phi|_{\mathrm{dom}(\phi)\setminus X}$ and $\phi'(x) \in \phi(T_x)$ for all $x \in X$.

Definition 8 (Instance of SPP)**.** Let $\mathcal{S}$ be an SPP $(\hat{s}\,\hat{\mathbf{x}}, \Delta)$ and ϕ a compatible valuation. The *(ϕ-)instance of $\mathcal{S}$*, denoted by $[\![\mathcal{S}]\!]_\phi$, is a pair $(\hat{s}(\phi \circ \hat{\mathbf{x}}), R)$, where R is the set of all triples $(s(\phi' \circ \mathbf{x}), c(\phi' \circ \mathbf{y}), s'(\phi' \circ \mathbf{x}'))$ such that $(s\,\mathbf{x}, X, \mathcal{C}, c\,\mathbf{y}, s'\mathbf{x}') \in \Delta$, $\phi' \in \mathrm{ext}(\phi, \mathrm{im}(\mathbf{x}) \cup X)$ and $[\![\mathcal{C}]\!]_{\phi'}$ is true.

In order to formalise the specification of HCP, we first capture it from the viewpoint of two hosts in an SPP $UA2$ in Figure 1, where a parametrised event $ih(z_h, z_a)$ denotes that a host z_h has an address z_a, guards other than $\top$ are written in brackets, and non-empty choice sets $\{x_1, \ldots, x_n\}$ are expressed in the form $\Box\, x_1, \ldots, x_n$ separated by a colon. In $UA2$, variables x_h and y_h of the type T_H representing hosts are free and variables x_a and y_a of the type T_A representing addresses are bound. That is why $\mathrm{par}(UA2) = \{x_h, y_h, T_A\}$ and $\overline{\mathrm{par}}(UA2) = \{x_h, y_h, T_A, T_H\}$. Initially, $UA2$ allows the host y_h to report having any address but after the host x_h has picked an address x_a, the host y_h is no longer allowed to report having x_a.

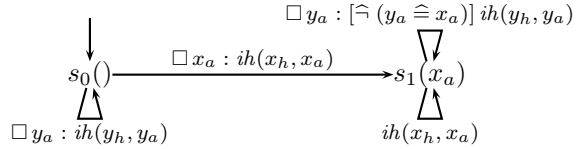

Figure 1. SPP $UA2$ representing the specification of HCP from the viewpoint of two hosts

Concurrent parametrised processes are constructed from SPPs, guards and parametrised versions of the operators.

Definition 9 (CPP)**.** *Concurrent parametrised processes (CPPs)* $\mathcal{P}$ are determined by the grammar:

$$\mathcal{P} ::= \mathcal{S} \mid ([\mathcal{C}]\mathcal{P}) \mid (\mathcal{P} \mathbin{\widehat{[\![}} \mathcal{P}) \mid (\mathcal{P} \mathbin{\widehat{\setminus}} E) \mid (\widehat{\|}_x \mathcal{P}),$$

where $\mathcal{S}$ ranges over SPPs, $\mathcal{C}$ over guards, E over the finite sets of visible channels and x over process variables.

A process variable x is *bound* in a CPP $\mathcal{P}$ if it occurs in $\mathcal{P}$ and its every occurrence is within a structure of the form $\widehat{\|}_y \mathcal{P}'$ such that $y = x$. A data variable x is *bound* in $\mathcal{P}$ if it occurs in $\mathcal{P}$ and it is not free in any SPP within $\mathcal{P}$. The other variables occurring in $\mathcal{P}$ are *free* in $\mathcal{P}$. The *parameters* of $\mathcal{P}$, the sets $\mathrm{par}(\mathcal{P})$ and $\overline{\mathrm{par}}(\mathcal{P})$, and the notion of compatibility are defined like for SPPs. We can also write $\mathcal{P}(x_1, \ldots, x_m, T_1, \ldots, T_n)$ to point out that $\mathcal{P}$ is a CPP the parameters of which are variables $x_1, \ldots, x_m$ and types $T_1, \ldots, T_n$.

Intuitively, each CPP represents (infinitely) many processes obtained by fixing the values of the parameters and evaluating the operators. Especially, $\widehat{\|}_x \mathcal{P}'$ denotes the parallel composition of all structures $\mathcal{P}'$ obtained by letting the variable x to range over its domain.

Definition 10 (Instance of CPP)**.** Let $\mathcal{P}$ be a CPP and ϕ a compatible valuation. The *(ϕ-)instance of $\mathcal{P}$*, denoted by $[\![\mathcal{P}]\!]_\phi$, is determined inductively as follows:

1) $[\![\mathcal{P}_1 \mathbin{\widehat{[\![}} \mathcal{P}_2]\!]_\phi = [\![\mathcal{P}_1]\!]_\phi \parallel [\![\mathcal{P}_2]\!]_\phi$,
2) $[\![\mathcal{P}' \setminus E]\!]_\phi = [\![\mathcal{P}']\!]_\phi \setminus E$,
3) $[\![[\mathcal{C}]\mathcal{P}']\!]_\phi = \begin{cases} [\![\mathcal{P}']\!]_\phi, & \text{if } [\![\mathcal{C}]\!]_\phi \text{ is } \mathit{true}, \\ P_{id}, & \text{if } [\![\mathcal{C}]\!]_\phi \text{ is } \mathit{false}, \text{ and} \end{cases}$
4) $[\![\widehat{\|}_x \mathcal{P}']\!]_\phi = \|_{\phi' \in \mathrm{ext}(\phi, \{x\})} [\![\mathcal{P}']\!]_{\phi'}$.

The ϕ-instance of a CPP $\mathcal{P}(x_1, \ldots, x_m, T_1, \ldots, T_n)$ is denoted by $\mathcal{P}(\phi(x_1), \ldots, \phi(x_m), \phi(T_1), \ldots, \phi(T_n))$, too. Nevertheless, $[\![\mathcal{P}]\!]_\phi$ is obviously a process with finitely many transitions and $[\![\mathcal{P}]\!]_\phi = [\![\mathcal{P}]\!]_\psi$ whenever ψ is a valuation such that $\psi|_{\mathrm{par}(\mathcal{P})} = \phi|_{\mathrm{par}(\mathcal{P})}$.

The model of the full specification is obtained by letting x_h and y_h to range over all pairs of different host identifiers and by composing all the resulting instances of $UA2$ in parallel. Hence, the formal specification is a CPP

$$UnqAdr(T_H, T_A) := \widehat{\|}_{x_h} \widehat{\|}_{y_h} [\widehat{\neg}\ (x_h \mathrel{\widehat{=}} y_h)]\, UA2\ ,$$

which allows for each host to report only a single unique address. To see this, consider a valuation ϕ such that $\phi(T_H) = \{h_1, h_2\}$ and $\phi(T_A) = A_2$, where $A_2 = \{a_1, a_2\}$. Obviously, ϕ is compatible with $UnqAdr$ and the ϕ-instance of $UnqAdr$ is the process

$$P_{id} \parallel (UA2(h_1, h_2, A_2) \parallel (UA2(h_2, h_1, A_2) \parallel P_{id}))$$

which is equal to $UA2(h_1, h_2, A_2) \parallel UA2(h_2, h_1, A_2)$ in Figure 2, modulo the structure of control states.

The protocol itself is modelled in a similar way and captured in a CPP

$$HCP(T_H, T_A) := \big(\, \widehat{\|}_{x_h} \widehat{\|}_{y_h} [\widehat{\neg}\ (x_h \mathrel{\widehat{=}} y_h)] Host\big) \mathbin{\widehat{\setminus}} IC,$$

where $Host$ is an SPP (in Figure 3 in [24]) and IC is the set of all visible channels occurring in $Host$, except for ih.

Refinement: We complete our parametrised formalism by defining a trace refinement relation on the set of CPPs.

Definition 11 (Parametrised Trace Refinement)**.** A CPP $\mathcal{P}_1$ is a *trace refinement* of a CPP $\mathcal{P}_2$, denoted by $\mathcal{P}_1 \mathrel{\widehat{\preceq}}_{\mathrm{tr}} \mathcal{P}_2$, if and only if $[\![\mathcal{P}_1]\!]_\phi \preceq_{\mathrm{tr}} [\![\mathcal{P}_2]\!]_\phi$ for all valuations ϕ compatible with both $\mathcal{P}_1$ and $\mathcal{P}_2$.

Given a system implementation CPP $\mathcal{P}$ and a system specification CPP $\mathcal{Q}$, we consider $\mathcal{P}$ to be *correct (with respect*

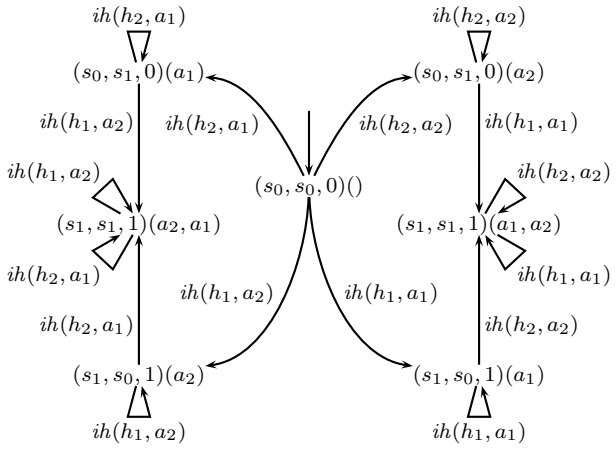

Figure 2. The specification of HCP with two hosts and two addresses

to $\mathcal{Q}$) if and only if $\mathcal{P} \mathrel{\widehat{\preceq}}_{\mathrm{tr}} \mathcal{Q}$. Now, the question on the correctness of HCP can be formalised as the problem whether $HCP \mathrel{\widehat{\preceq}}_{\mathrm{tr}} UnqAdr$. Like the original trace refinement, also its parametrised version allows for compositional analysis.

Proposition 12. *The relation $\mathrel{\widehat{\preceq}}_{\mathrm{tr}}$ is a precongruence on the set of CPPs.*

IV. Cut-Off Theorems

Next, we present the main result of the paper, Corollary 21, which provides cut-offs for checking trace refinement between CPPs. However, we need to assume that the specification does not involve hiding and that it is *deterministic*, i.e., all its instances are deterministic. That is because giving up either of the assumptions renders parametrised verification undecidable.[1] Fortunately, in practice, many safety specifications are deterministic and hiding is typically applied only on the implementation side.

The proof consists of two main parts, Theorems 16 and 20, which give cut-offs for process types and data types, respectively. Theorem 16 is similar to the ones proved in [7], [5] but allows for data parameters, too, whereas Theorem 20 is semantically similar to the DI results [8], [9] but technically more elegant. Consequently, the presentation is more accessible and concise here than in [8], [9].

Finally, we provide cut-offs for checking the determinism of a CPP in the form of Theorem 22. Previously, a semantically similar result is presented in [25] in the context of [8].

[1]To see why allowing hiding on the specification side makes parametrised verification undecidable, we refer to [5]. The undecidability for non-deterministic specifications follows from the fact that they can detect whether an implementation uses data constants for counting. As long as the implementation maintains the initial order for any (non-deterministically chosen) pair of data constants, the specification does not allow for an extra behaviour, especially executing a halting event is forbidden. However, if a pair of data constants does not respect the order in which they were seen for the first time, the specification allows the implementation to do anything. This way, the implementation can simulate a Turing machine and halting can be detected as the violation of the specification.

A. Cut-Offs for Process Types

In order to determine cut-offs for process types, we first show that if a big instance of the system specification $\mathcal{Q}$ is composed of the same components as a set of small instances, then the big instance can be represented as the parallel composition of the small ones. (Here, big and small are in terms of the size of the values of process types.) Similarly, if a big instance of the system implementation $\mathcal{P}$ is composed of the same components as a set of small instances, the big instance can be over-approximated as the parallel composition of the small ones (Lemma 13).

Second, if each of the small instances of $\mathcal{P}$ is a trace refinement of the corresponding instance of $\mathcal{Q}$, then by the compositionality of the trace refinement, the parallel composition of the small instances of $\mathcal{P}$ is a trace refinement of the parallel composition of the small instances of $\mathcal{Q}$. By above and the transitivity of the trace refinement, it implies that the big instance of $\mathcal{P}$ is a trace refinement of the corresponding instance of $\mathcal{Q}$, too (Proposition 14).

Finally, since there is an upper limit for the size of small instances which is obtained by simply counting the number of free and nested bound variables for each process type (Lemma 15), a parametrised trace refinement checking task reduces to the refinement checking of the instances, where the size of the process types is bounded (Theorem 16).

In order to present the technique formally, we need to clarify some concepts. If $\mathcal{R}$ is a CPP and ϕ a compatible valuation, *the set of the processes (of the ϕ-instance of $\mathcal{R}$)*, denoted by $\mathrm{prc}(\mathcal{R}, \phi)$, is defined inductively as follows:

1) $\mathrm{prc}(\mathcal{S}, \phi) = \{[\![\mathcal{S}]\!]_\phi\}$ for an SPP $\mathcal{S}$,
2) $\mathrm{prc}(\mathcal{R}_1 \mathbin{\widehat{\|}} \mathcal{R}_2, \phi) = \bigcup_{i \in \{1,2\}} \big(\{i\} \times \mathrm{prc}(\mathcal{R}_i, \phi)\big)$,
3) $\mathrm{prc}(\mathcal{R} \mathbin{\widehat{\setminus}} E, \phi) = \mathrm{prc}(\mathcal{R}, \phi)$,
4) $\mathrm{prc}([\mathcal{C}]\mathcal{R}', \phi) = \begin{cases} \mathrm{prc}(\mathcal{R}', \phi), & \text{if } [\![\mathcal{C}]\!]_\phi \text{ is } \mathit{true}, \\ \emptyset, & \text{if } [\![\mathcal{C}]\!]_\phi \text{ is } \mathit{false}, \end{cases}$ and
5) $\mathrm{prc}(\widehat{\|}_x \mathcal{R}', \phi) = \bigcup_{\phi' \in \mathrm{ext}(\phi, \{x\})} \big(\{\phi'(x)\} \times \mathrm{prc}(\mathcal{R}', \phi')\big)$.

For example, if θ is a valuation such that $T_H, T_A \in \mathrm{dom}(\theta)$ and $\theta(T_H) = \{h_1, \ldots, h_n\}$, then $\mathrm{prc}(UnqAdr, \theta)$ equals

$$\big\{(h_i, (h_j, UA2(h_i, h_j, \theta(T_A)))) \mid i, j \in \{1, \ldots, n\}, i \neq j\big\} .$$

If ϕ_1 and ϕ_2 are valuations and $\mathcal{T}$ is a set of types, we say that ϕ_1 is a *($\mathcal{T}$-)subvaluation* of ϕ_2, if

1) ϕ_1 and ϕ_2 have the same domain,
2) $\phi_1(T) \subseteq \phi_2(T)$ for all types $T \in \mathcal{T} \cap \mathrm{dom}(\phi_1)$ and
3) $\phi_1|_{\mathrm{dom}(\phi_1) \setminus \mathcal{T}} = \phi_2|_{\mathrm{dom}(\phi_2) \setminus \mathcal{T}}$ (the valuations agree on the values of parameters outside $\mathcal{T}$).

The ϕ_1-instance of a CPP $\mathcal{R}$ is *smaller than (or equal to)* the ϕ_2-instance of $\mathcal{R}$ if ϕ_1 is a subvaluation of ϕ_2. For example, if Θ is the set of all subvaluations θ' of θ such that $|\theta'(T_H)| \leq 2$, then Θ is a finite set of $\mathbb{T}_P$-subvaluations of θ and $[\![UnqAdr]\!]_{\theta'}$ is smaller than $[\![UnqAdr]\!]_\theta$ for all $\theta' \in \Theta$.

Since the specification $\mathcal{Q}$ does not involve hiding, each instance of $\mathcal{Q}$ is just the parallel composition of the instances

of SPPs occurring $\mathcal{Q}$. Hence, if the set of the processes of a (big) instance of $\mathcal{Q}$ equals the set of the processes of smaller instances of $\mathcal{Q}$, then by the commutativity, associativity and idempotence of the parallel composition, it is evident that the big instance is trace equivalent to the parallel composition of the small instances. For implementation CPPs $\mathcal{P}$, which typically involve hiding, trace equivalence does not hold in general. However, since distributing hiding over the parallel composition results in a process with more traces, we can still establish a trace refinement between the big instance of $\mathcal{P}$ and the parallel composition of the small ones.

Lemma 13. *Let $\mathcal{R}$ be a CPP, ψ a compatible valuation, $T \in \mathrm{dom}(\psi)$ a process type and Φ a finite set of the $\{T\}$-subvaluations of ψ such that $\mathrm{prc}(\mathcal{R},\psi) = \bigcup_{\phi\in\Phi}\mathrm{prc}(\mathcal{R},\phi)$. Then $[\![\mathcal{R}]\!]_\psi \preceq_{\mathrm{tr}} \|_{\phi\in\Phi}[\![\mathcal{R}]\!]_\phi$, and if $\mathcal{R}$ does not involve hiding, $[\![\mathcal{R}]\!]_\psi \equiv_{\mathrm{tr}} \|_{\phi\in\Phi}[\![\mathcal{R}]\!]_\phi$, too.*

Considering our running example, let θ and Θ be as above. Since every element of $\mathrm{prc}(\mathit{UnqAdr},\theta)$ depends on the identifiers of two hosts, it is easy to see that $\mathrm{prc}(\mathit{UnqAdr},\theta)$ equals $\bigcup_{\theta'\in\Theta}\mathrm{prc}(\mathit{UnqAdr},\theta')$, which implies that $[\![\mathit{UnqAdr}]\!]_\theta \equiv_{\mathrm{tr}} \|_{\theta'\in\Theta}[\![\mathit{UnqAdr}]\!]_{\theta'}$. Similarly, we can check that $\mathrm{prc}(\mathit{HCP},\theta) = \bigcup_{\theta'\in\Theta}\mathrm{prc}(\mathit{HCP},\theta')$, which implies that $[\![\mathit{HCP}]\!]_\theta \preceq_{\mathrm{tr}} \|_{\theta'\in\Theta}[\![\mathit{HCP}]\!]_{\theta'}$. Now, if $[\![\mathit{HCP}]\!]_{\theta'} \preceq_{\mathrm{tr}} [\![\mathit{UnqAdr}]\!]_{\theta'}$ for all $\theta' \in \Theta$, then by the compositionality of $\preceq_{\mathrm{tr}}$, we know that $\|_{\theta'\in\Theta}[\![\mathit{HCP}]\!]_{\theta'} \preceq_{\mathrm{tr}} \|_{\theta'\in\Theta}[\![\mathit{UnqAdr}]\!]_{\theta'}$, too. By the transitivity of $\preceq_{\mathrm{tr}}$, it follows that $[\![\mathit{HCP}]\!]_\theta \preceq_{\mathrm{tr}} [\![\mathit{UnqAdr}]\!]_\theta$. Hence, the lemma implies that we can derive the correctness of a big system instance from the correctness of small instances.

Proposition 14. *Let $\mathcal{P}$ and $\mathcal{Q}$ be CPPs such that $\mathcal{Q}$ does not involve hiding, ψ a compatible valuation, $T \in \mathrm{dom}(\psi)$ a process type and Φ a set of the $\{T\}$-subvaluations of ψ such that $\mathrm{prc}(\mathcal{P},\psi) = \bigcup_{\phi\in\Phi}\mathrm{prc}(\mathcal{P},\phi)$ and $\mathrm{prc}(\mathcal{Q},\psi) = \bigcup_{\phi\in\Phi}\mathrm{prc}(\mathcal{Q},\phi)$. If $[\![\mathcal{P}]\!]_\phi \preceq_{\mathrm{tr}} [\![\mathcal{Q}]\!]_\phi$ for all valuations $\phi \in \Phi$, then $[\![\mathcal{P}]\!]_\psi \preceq_{\mathrm{tr}} [\![\mathcal{Q}]\!]_\psi$.*

The proposition allows us to discard (big) instances but it does not explicitly say which instances we should keep. This piece of information is hidden in the condition of the form $\mathrm{prc}(\mathcal{R},\psi) = \bigcup_{\phi\in\Phi}\mathrm{prc}(\mathcal{R},\phi)$ which requires the set of the processes of the big instance to be the same as the set of the processes of the small ones. Since each element in $\mathrm{prc}(\mathcal{R},\psi)$ is completely determined by the values of finitely many variables and data types, the *cut-off (size)* for a process type T and $\mathcal{R}$, denoted by $\mathrm{cp}_T(\mathcal{R})$, depends on the maximum number of constants in I_T that occur in an element in $\mathrm{prc}(\mathcal{R},\psi)$. In other words, $\mathrm{cp}_T(\mathcal{R})$ is obtained by simply counting the number of the free variables of the type T and adding the maximum number of the nested bound variables of the type T. More formally,

$$\mathrm{cp}_T(\mathcal{R}) := \max(1, |\mathrm{free}_T(\mathcal{R})| + \mathrm{pbnd}_T(\mathcal{R}))\ ,$$

where $\mathrm{free}_T(\mathcal{R})$ is the set $\{x \in \mathbb{X} \cap \mathrm{par}(\mathcal{R}) \mid T_x = T\}$ of all the variables of the type T that are free in $\mathcal{R}$ and $\mathrm{pbnd}_T(\mathcal{R})$ is defined inductively as follows:

1) $\mathrm{pbnd}_T((\hat{s}\,\hat{\mathbf{x}},\Delta)) = 0$,
2) $\mathrm{pbnd}_T([\mathcal{C}]\mathcal{R}') = \mathrm{pbnd}_T(\mathcal{R}')$,
3) $\mathrm{pbnd}_T(\mathcal{R}_1 \mathbin{\widehat{\|}} \mathcal{R}_2) = \max(\mathrm{pbnd}_T(\mathcal{R}_1), \mathrm{pbnd}_T(\mathcal{R}_2))$,
4) $\mathrm{pbnd}_T(\mathcal{R}' \setminus E) = \mathrm{pbnd}_T(\mathcal{R}')$ and
5) $\mathrm{pbnd}_T(\widehat{\|}_x \mathcal{R}') = \begin{cases} \mathrm{pbnd}_T(\mathcal{R}') + 1, & \text{if } T_x = T, \\ \mathrm{pbnd}_T(\mathcal{R}'), & \text{if } T_x \neq T. \end{cases}$

For example, as the specification CPP UnqAdr involves two subprocesses of the form $\widehat{\|}_x \mathcal{P}'$ such that $T_x = T_H$, we see that $\mathrm{pbnd}_{T_H}(\mathit{UnqAdr}) = 2$. Moreover, since UnqAdr has no free variable, it implies that $\mathrm{cp}_{T_H}(\mathit{UnqAdr}) = 2$. Similarly, we can show that $\mathrm{cp}_{T_H}(\mathit{HCP}) = 2$, too.

Lemma 15. *If $\mathcal{R}$ is a CPP, ψ a compatible valuation, $T \in \mathrm{dom}(\psi)$ a process type, $k \geq |\mathrm{im}(\psi|_{\mathbb{X}}) \cap I_T| + \mathrm{pbnd}_T(\mathcal{R})$ a positive integer and Φ the set of all $\{T\}$-subvaluations ϕ of ψ such that $|\phi(T)| = \min(k, |\psi(T)|)$, then $\mathrm{prc}(\mathcal{R},\psi) = \bigcup_{\phi\in\Phi}\mathrm{prc}(\mathcal{R},\phi)$.*

By combining the results above, we get cut-offs for process types for parametrised trace refinement checking.

Theorem 16 (Cut-Offs for Process Types)**.** *Let $\mathcal{P}$ and $\mathcal{Q}$ be CPPs such that $\mathcal{Q}$ does not involve hiding, Φ the set of all valuations with the domain $\overline{\mathrm{par}}(\mathcal{P} \mathbin{\widehat{\|}} \mathcal{Q})$ and Ψ the set of all valuations $\phi \in \Phi$ such that $|\phi(T)| \leq \mathrm{cp}_T(\mathcal{P} \mathbin{\widehat{\|}} \mathcal{Q})$ for every process type $T \in \overline{\mathrm{par}}(\mathcal{P} \mathbin{\widehat{\|}} \mathcal{Q})$. Then $[\![\mathcal{P}]\!]_\phi \preceq_{\mathrm{tr}} [\![\mathcal{Q}]\!]_\phi$ for all $\phi \in \Phi$, if and only if $[\![\mathcal{P}]\!]_\psi \preceq_{\mathrm{tr}} [\![\mathcal{Q}]\!]_\psi$ for all $\psi \in \Psi$.*

When we apply the theorem to HCP, we see that to prove HCP correct for any number of hosts and network addresses, it is sufficient to consider instances with an arbitrary number of addresses but at most two hosts.

B. Cut-Offs for Data Types

To determine cut-offs for data types, we first show that an execution of a big instance of a system implementation $\mathcal{P}$ can be bijectively mapped to an execution of a small instance of $\mathcal{P}$ transition by transition. (Here, big and small are in terms of the size of the values of data types.) If the small instance of $\mathcal{P}$ is a trace refinement of the small instance of the system specification $\mathcal{Q}$, then the small specification instance has a corresponding execution which can be projected back to the execution of the big instance of $\mathcal{Q}$ by using the inverse mappings, provided the specification is deterministic (Lemma 17). This implies that if the small instance of $\mathcal{P}$ is a trace refinement of the small instance $\mathcal{Q}$, then the big instance of $\mathcal{P}$ is a trace refinement of the big instance $\mathcal{Q}$, too (Prop. 18).

Since we can compute an upper limit for the size of small instances based on the cut-offs of process types and the number of data variables occurring in $\mathcal{P}$ and $\mathcal{Q}$ (Lemma 19), a parametrised trace refinement checking task reduces to the refinement checking of the instances of bounded size (Theorem 20).

The bijective mapping of executions between big and small instances presumes that the small instance is big enough. The sufficient size for the small instance depends on the number of constants occurring in a transition or the initial state of the big instance, which motivates the following definition. Let P be a process and A a set of constants. We write $\mathrm{idc}(P, A)$ for the maximum number of constants in A that occur in a state or transition of P, i.e.,

$$\mathrm{idc}(P, A) := \max\{|\mathrm{im}(\hat{\mathbf{d}}) \cap A|, |\mathrm{im}(\mathbf{d}\mathbf{e}\mathbf{d}') \cap A| \mid \hat{s}\,\hat{\mathbf{d}} = \mathrm{init}(P), (s\,\mathbf{d}, c\,\mathbf{e}, s'\mathbf{d}') \in \mathrm{tran}(P)\} .$$

For example, let θ_1 be a valuation with the domain $\{T_A, T_H\}$ such that $\theta_1(T_H) = \{h_1, h_2\}$ and $\theta_1(T_A) = \{a_1, \ldots, a_n\}$. If $n = 2$, then $[\![\mathit{UnqAdr}]\!]_{\theta_1}$ is the process in Figure 2, where obviously $\mathrm{idc}([\![\mathit{UnqAdr}]\!]_{\theta_1}, I_{T_A}) = 2$. More generally, since $[\![\mathit{UnqAdr}]\!]_{\theta_1} = P_{id} \parallel \mathit{UA2}(h_1, h_2, \theta_1(T_A)) \parallel \mathit{UA2}(h_2, h_1, \theta_1(T_A)) \parallel P_{id}$ and each parametrised transition and state of $\mathit{UA2}$ involves at most two variables of the type T_A, we can see that $\mathrm{idc}([\![\mathit{UnqAdr}]\!]_{\theta_1}, I_{T_A}) \leq 4$. Similarly, we get that $\mathrm{idc}([\![\mathit{HCP}]\!]_{\theta_1}, I_{T_A}) \leq 4$.

Lemma 17. *Let $\mathcal{P}$ and $\mathcal{Q}$ be CPPs, ψ a compatible valuation, $T \in \mathrm{dom}(\psi)$ a data type and ϕ a $\{T\}$-subvaluation of ψ such that $|\phi(T)| \geq |\mathrm{im}(\psi|_{\mathbb{X}}) \cap I_T| + \mathrm{idc}\big([\![\mathcal{P}]\!]_\psi, I_T \setminus \mathrm{im}(\psi|_{\mathbb{X}})\big) + \mathrm{idc}\big([\![\mathcal{Q}]\!]_\phi, I_T \setminus \mathrm{im}(\psi|_{\mathbb{X}})\big)$. Moreover, let $[\![\mathcal{Q}]\!]_\phi$ be deterministic, $[\![\mathcal{P}]\!]_\phi$ a trace refinement of $[\![\mathcal{Q}]\!]_\phi$ and $(s_0\,\mathbf{d}_0, c_1\,\mathbf{e}_1, s_1\,\mathbf{d}_1, \ldots, c_n\,\mathbf{e}_n, s_n\,\mathbf{d}_n)$ an execution of $[\![\mathcal{P}]\!]_\psi$ such that $n = 0$ or $c_n\,\mathbf{e}_n$ is a visible event, and let $i_1, i_2, \ldots, i_k$ be the increasing sequence of all the indices in $\{1, \ldots, n\}$ such that $c_i \neq \tau$.*

Then, (i) there are bijections $g_1, \ldots, g_n : \mathbb{V} \mapsto \mathbb{V}$ which move only constants in $\psi(T) \setminus \mathrm{im}(\psi|_{\mathbb{X}})$ such that $(s_0\,\mathbf{d}_0, c_1(g_1 \circ \mathbf{e}_1), s_1(g_1 \circ \mathbf{d}_1), \ldots, c_n(g_n \circ \mathbf{e}_n), s_n(g_n \circ \mathbf{d}_n))$ is an execution of $[\![\mathcal{P}]\!]_\phi$, (ii) and there is an execution $(s'_0\,\mathbf{d}'_0, c'_1\,\mathbf{e}'_1, s'_1\,\mathbf{d}'_1, \ldots, c'_k\,\mathbf{e}'_k, s'_k\,\mathbf{d}'_k)$ of $[\![\mathcal{Q}]\!]_\phi$ such that $(s'_0\mathbf{d}'_0, c'_1(g_{i_1}^{-1} \mathbin{\circ} \mathbf{e}'_1), s'_1(g_{i_1}^{-1} \mathbin{\circ} \mathbf{d}'_1), \ldots, c'_k(g_{i_k}^{-1} \mathbin{\circ} \mathbf{e}'_k), s'_k(g_{i_k}^{-1} \mathbin{\circ} \mathbf{d}'_k))$ is an execution of $[\![\mathcal{Q}]\!]_\psi$ and $c'_j(g_{i_j}^{-1} \circ \mathbf{e}'_j) = c_{i_j}\,\mathbf{e}_{i_j}$ for all $j \in \{1, \ldots, k\}$.

Regarding HCP, let θ_2 be a $\{T_A\}$-subvaluation of θ_1 such that $|\theta_2(T_A)| = 8$. Since $[\![\mathit{UnqAdr}]\!]_{\theta_2}$ is deterministic, $\mathrm{im}(\theta_1|_{\mathbb{X}})$ is empty and $|\theta_2(T_A)| \geq \mathrm{idc}\big([\![\mathit{HCP}]\!]_{\theta_1}, I_{T_A}\big) + \mathrm{idc}\big([\![\mathit{UnqAdr}]\!]_{\theta_2}, I_{T_A}\big)$, the proposition implies that if $[\![\mathit{HCP}]\!]_{\theta_2} \preceq_{\mathrm{tr}} [\![\mathit{UnqAdr}]\!]_{\theta_2}$, then for every execution of $[\![\mathit{HCP}]\!]_{\theta_1}$ there is an execution of $[\![\mathit{UnqAdr}]\!]_{\theta_1}$ which gives rise to the same trace. Hence, with the aid of the lemma above, we can derive the correctness of a big implementation instance from the correctness of a small one.

Proposition 18. *Let $\mathcal{P}$ and $\mathcal{Q}$ be CPPs, ψ a compatible valuation, $T \in \mathrm{dom}(\psi)$ a data type and ϕ a $\{T\}$-subvaluation of ψ such that $|\phi(T)| \geq |\mathrm{im}(\psi|_{\mathbb{X}}) \cap I_T| + \mathrm{idc}\big([\![\mathcal{P}]\!]_\psi, I_T \setminus \mathrm{im}(\psi|_{\mathbb{X}})\big) + \mathrm{idc}\big([\![\mathcal{Q}]\!]_\psi, I_T \setminus \mathrm{im}(\psi|_{\mathbb{X}})\big)$. If $[\![\mathcal{Q}]\!]_\phi$ is deterministic and $[\![\mathcal{P}]\!]_\phi \preceq_{\mathrm{tr}} [\![\mathcal{Q}]\!]_\phi$, then $[\![\mathcal{P}]\!]_\psi \preceq_{\mathrm{tr}} [\![\mathcal{Q}]\!]_\psi$, too.*

Like in the case of process types, the proposition allows us to discard (big) instances but it does not explicitly say which instances we should keep. This time, the cut-off size of a data type T depends on the number of constants in I_T that occur in a state or transition of an implementation instance and a specification instance. If the size of process types is already bounded using Theorem 16, then the *cut-off (size)* for T and a CPP $\mathcal{R}$, denoted by $\mathrm{cd}_T(\mathcal{R})$, can be over-approximated as follows. First, count the number of the free variables of the type T. Then, go through all SPPs $\mathcal{S}$ occurring in $\mathcal{R}$ and in each round, add the product of two figures: the maximum number of the bound variables of the type T occurring in a parametrised transition and the maximum number of the instances of $\mathcal{S}$ within an instance of $\mathcal{R}$. To put it more formally,

$$\mathrm{cd}_T(\mathcal{R}) := \max(1, |\mathrm{free}_T(\mathcal{R})| + \mathrm{dbnd}_{T,\mathcal{R}}(\mathcal{R})) ,$$

where $\mathrm{dbnd}_{T,\mathcal{R}}(\mathcal{R})$ is defined inductively as follows:

1) $\mathrm{dbnd}_{T,\mathcal{R}}((\hat{s}\,\hat{\mathbf{x}}, \Delta))$ is the maximum of $|\{x \in \mathrm{im}(\mathbf{x}) \cup X \mid T_x = T\}|$ when $(s\,\mathbf{x}, X, \mathcal{C}, c\,\mathbf{y}, s'\mathbf{x}')$ ranges over Δ,
2) $\mathrm{dbnd}_{T,\mathcal{R}}([\mathcal{C}]\mathcal{R}') = \mathrm{dbnd}_{T,\mathcal{R}}(\mathcal{R}')$,
3) $\mathrm{dbnd}_{T,\mathcal{R}}(\mathcal{R}_1 \mathbin{\widehat{\|}} \mathcal{R}_2) = \mathrm{dbnd}_{T,\mathcal{R}}(\mathcal{R}_1) + \mathrm{dbnd}_{T,\mathcal{R}}(\mathcal{R}_2)$,
4) $\mathrm{dbnd}_{T,\mathcal{R}}(\mathcal{R}' \mathbin{\widehat{\setminus}} E) = \mathrm{dbnd}_{T,\mathcal{R}}(\mathcal{R}')$ and
5) $\mathrm{dbnd}_{T,\mathcal{R}}(\widehat{\|}_x \mathcal{R}') = \mathrm{cp}_{T_x}(\mathcal{R}) \cdot \mathrm{dbnd}_{T,\mathcal{R}}(\mathcal{R}')$.

For example, let $\mathit{HU} := \mathit{HCP} \mathbin{\widehat{\|}} \mathit{UnqAdr}$. By above, we know that $\mathrm{cp}_{T_H}(\mathit{HU}) = 2$. Moreover, since every parametrised transition of $\mathit{UA2}$ involves at most two variables of the type T_A, we see that $\mathrm{dbnd}_{T_A,\mathit{HU}}(\mathit{UA2}) = 2$. Hence, $\mathrm{dbnd}_{T_A,\mathit{HU}}(\mathit{UnqAdr}) = \mathrm{cp}_{T_{x_h}}(\mathit{HU}) \cdot \mathrm{cp}_{T_{x_y}}(\mathit{HU}) \cdot \mathrm{dbnd}_{T_A,\mathit{HU}}(\mathit{UA2}) = 8$. Similarly, we can show that $\mathrm{dbnd}_{T_A,\mathit{HU}}(\mathit{HCP}) = 8$. Because HU has no free variable, it implies that $\mathrm{cd}_{T_A}(\mathit{HU}) = 16$.

Lemma 19. *Let $\mathcal{P}$ be a CPP, ϕ a compatible valuation and $\mathcal{R}$ a CPP such that $|\phi(U)| \leq \mathrm{cp}_U(\mathcal{R})$ for each process type $U \in \mathrm{dom}(\phi)$. Then for every data type $T \in \mathrm{dom}(\phi)$, $\mathrm{dbnd}_{T,\mathcal{R}}(\mathcal{P}) \geq \mathrm{idc}\big([\![\mathcal{P}]\!]_\phi, I_T \setminus \mathrm{im}(\psi|_{\mathbb{X}})\big)$.*

By combining the results above, we get cut-offs for data types for parametrised trace refinement checking.

Theorem 20 (Cut-Offs for Data Types)**.** *Let $\mathcal{P}$ and $\mathcal{Q}$ be CPPs such that $\mathcal{Q}$ is deterministic. Moreover, let Ψ be the set of all valuations ψ with the domain $\overline{\mathrm{par}}(\mathcal{P} \mathbin{\widehat{\|}} \mathcal{Q})$ such that $|\psi(U)| \leq \mathrm{cp}_U(\mathcal{P} \mathbin{\widehat{\|}} \mathcal{Q})$ for every process type $U \in \overline{\mathrm{par}}(\mathcal{P} \mathbin{\widehat{\|}} \mathcal{Q})$, and Φ the set of all valuations $\phi \in \Psi$ such that $|\phi(T)| \leq \mathrm{cd}_T(\mathcal{P} \mathbin{\widehat{\|}} \mathcal{Q})$ for every data type $T \in \overline{\mathrm{par}}(\mathcal{P} \mathbin{\widehat{\|}} \mathcal{Q})$. Then, $[\![\mathcal{P}]\!]_\psi \preceq_{\mathrm{tr}} [\![\mathcal{Q}]\!]_\psi$ for all $\psi \in \Psi$, if and only if $[\![\mathcal{P}]\!]_\phi \preceq_{\mathrm{tr}} [\![\mathcal{Q}]\!]_\phi$ for all $\phi \in \Phi$.*

When we apply the theorem to HCP, we see that to prove the system correct for any number of addresses and up to two hosts, it is sufficient to consider instances with at most 16 addresses and two hosts.

C. Automated Approach

Even though Theorems 16 and 20 provide cut-offs for types, they do not quite enable automated parametrised verification because there are still infinitely many instances up to the cut-offs. However, if we consider only valuations which are *non-isomorphic*, i.e., cannot be obtained from each other by the bijective mapping of constants, then we are left with finitely many instances. Formally, valuations ϕ_1 and ϕ_2 are *isomorphic*, if they have the same domain and if there is a bijection $g : \mathbb{V} \mapsto \mathbb{V}$ such that $\phi_2(x) = g(\phi_1(x))$ for all variables $x \in \mathrm{dom}(\phi_1)$ and $\phi_2(T) = \{g(c) \mid c \in \phi_1(T)\}$ for all types $T \in \mathrm{dom}(\phi_1)$. Valuations are *non-isomorphic* if they are not isomorphic.

Getting rid of isomorphs is safe since isomorphic valuations result in verification tasks with the same answer. That why we say that a set Φ of valuations is a *cut-off set for* a CPP $\mathcal{R}$ if Φ is a maximal set of non-isomorphic valuations ϕ with the domain $\overline{\mathrm{par}}(\mathcal{R})$ such that $|\phi(U)| \leq \mathrm{cp}_U(\mathcal{R})$ for every process type $U \in \mathrm{dom}(\phi)$ and $|\phi(T)| \leq \mathrm{cd}_T(\mathcal{R})$ for every data type $T \in \mathrm{dom}(\phi)$. With Theorems 16 and 20, this notion leads to our main result, which allow us to reduce a parametrised verification task to finitely many refinement checks between finite processes.

Corollary 21 (The Main Cut-Off Theorem)**.** *Let Φ be a cut-off set for a CPP $(\mathcal{P} \,\widehat{\|}\, \mathcal{Q})$, where $\mathcal{Q}$ is deterministic and does not involve hiding. Then (1) Φ is finite and (2) $\mathcal{P} \,\widehat{\preceq}_{\mathrm{tr}}\, \mathcal{Q}$ if and only if $[\![\mathcal{P}]\!]_\phi \preceq_{\mathrm{tr}} [\![\mathcal{Q}]\!]_\phi$ for all $\phi \in \Phi$.*

Another practical problem is that establishing the determinism of the specification involves infinitely many checks, too. That is why we provide cut-offs for that task as well by using similar reasoning as above.

Theorem 22 (Cut-Offs for Determinism Checking)**.** *Let Φ be a cut-off set for a CPP $(\mathcal{Q} \,\widehat{\|}\, \mathcal{Q})$, where $\mathcal{Q}$ does not involve hiding. Then (1) Φ is finite and (2) $\mathcal{Q}$ is deterministic, if and only if $[\![\mathcal{Q}]\!]_\phi$ is deterministic for all $\phi \in \Phi$.*

Corollary 21 and Theorem 22 give rise to a completely automatic procedure for parametrised verification which we have implemented in the recent version of the Bounds tool [26]. First, the tool reads system implementation and specification CPPs. Then, it computes the cut-off sizes for types, produces the instances of the specification up to the cut-offs and checks that they are deterministic. Moreover, since the cut-offs provided by the corollary are only rough structural ones, the tool tries to improve them further by checking the assumptions of Propositions 14 and 18 for each instance up to the rough bounds and by discarding big instances which satisfy the assumptions of either proposition. After that, Bounds produces the trace refinement checking tasks up to the improved cut-offs. Finally, the outputted finite state verification questions are solved using the refinement checker FDR2 [3] in order to obtain the answer to the parametrised verification task.

When applied to HCP, Bounds first generates all 32 instances up to the cut-offs of two hosts and 16 addresses and checks that the specification is deterministic. After that, it applies Propositions 14 and 18, which reveals that only six of the instances have to be actually verified. By using FDR2, all six instances are found to be correct, which implies that HCP operates correctly for any number of hosts and addresses.

We have also proved mutual exclusion properties for a shared resource system (SRS) where an arbitrary number of users compete for an access to an arbitrary number of shared resources which store data from an arbitrary large domain and a cache coherence protocol with an arbitrary number of processors and an arbitrary amount of memory storing arbitrary large data values. Additionally, we have proved consistency properties for both systems when the number of users and processes was fixed to two. In each case, the whole verification process took only a second or two (see Table I), which shows that our approach is applicable to practical verification problems. Moreover, since our formalism is compositional, the specifications of these systems can be used in place of the implementations in further verification efforts. The experiments were run on a desktop computer with 4GB of memory running 64-bit Ubuntu on 3Ghz Intel Core2 Duo processor. The descriptions of HCP, SRS and Cache models are available in [24].

V. Conclusions and Future Work

We have presented a formalism for expressing parametrised systems and combined two existing results [8], [5] on parametrised verification into a single powerful technique which enables parametrised verification by reduction to finitely many finite state verification tasks. The reduction is determined by the structure of a system implementation and specification and, to the best of our knowledge, this is the first and only parametrised verification technique that allows for compositional reasoning, lends support to multiple and two fundamentally different kinds of parameters and guarantees termination with the correct answer. The approach is implemented in a tool used to prove the correctness of several parametrised systems.

An obvious topic for future research is extending the approach. We already know that the approach can be extended with relations over process types that are definable in the first order logic [7], [5] but we are also interested in mixing process and data types and extending the technique to liveness properties [10] and interface theories [27]. The challenge is that such extensions tend to render parametrised refinement checking undecidable [5]. Nevertheless, we are hopeful in finding practically sensible assumptions under which the extensions can be realised.

Acknowledgement: The work was partially funded by Helsinki Institute for Information Technology HIIT,

Table I
STATISTICS ON THE VERIFICATION OF SYSTEMS INVOLVING BOTH PARAMETRIC PROCESS AND DATA TYPES

	Process types		Data types		Instances		Processing time	
System	number	cut-offs	number	cut-offs	generated	outputted	Bounds	FDR2
HCP	1	2	1	16	32	6	0.20s	0.13s
SRS mutex	2	2,1	1	12	24	5	0.06s	0.11s
SRS consistency	1	1	1	8	8	6	0.04s	0.15s
Cache mutex	2	2,1	1	17	34	12	0.23s	0.35s
Cache consistency	1	1	1	14	14	12	0.28s	1.05s

Academy of Finland (project 139402), and RECOMP project funded by ARTEMIS-JU.

REFERENCES

[1] K. R. Apt and D. C. Kozen, "Limits for automatic verification of finite-state concurrent systems," *Inf. Process. Lett.*, vol. 22, no. 6, pp. 307–309, 1986.

[2] C. A. R. Hoare, *Communicating sequential processes*. Prentice-Hall, 1985.

[3] A. W. Roscoe, *Understanding Concurrent Systems*. Springer, 2010.

[4] A. Siirtola, "Automated multiparameterised verification by cut-offs," in *ICFEM 2010*, ser. LNCS. Springer, 2010, vol. 6447, pp. 321–337.

[5] ——, "Algorithmic multiparameterised verification of safety properties. Process algebraic approach," Ph.D. dissertation, University of Oulu, 2010.

[6] A. Siirtola and J. Kortelainen, "Parameterised process algebraic verification by precongruence reduction," in *ACSD '09*. IEEE, 2009, pp. 158–167.

[7] ——, "Algorithmic verification with multiple and nested parameters," in *ICFEM '09*, ser. LNCS. Springer, 2009, vol. 5885, pp. 561–580.

[8] R. S. Lazić, "A semantic study of data independence with applications to model checking," Ph.D. dissertation, Oxford University, 2001.

[9] R. S. Lazić and D. Nowak, "A unifying approach to data-independence," in *CONCUR '00*, ser. LNCS. Springer, 2000, vol. 1877, pp. 581–595.

[10] A. Valmari and M. Tienari, "An improved failures equivalence for finite-state systems with a reduction algorithm," in *PSTV '91*. North-Holland, 1991, pp. 3–18.

[11] S. J. Creese, "Data independent induction: CSP model checking of arbitrary sized networks," Ph.D. dissertation, Oxford University, 2001.

[12] S. Ghilardi and S. Ranise, "Backward reachability of array-based systems by SMT solving: termination and invariant synthesis," *Log. Meth. Comput. Sci.*, vol. 6, no. 4, 2010.

[13] A. Finkel and P. Schnoebelen, "Well-structured transition systems everywhere!" *Theor. Comput. Sci.*, vol. 256, no. 1, pp. 63–92, 2001.

[14] E. A. Emerson and V. Kahlon, "Reducing model checking of the many to the few," in *CADE-17*, ser. LNCS. Springer, 2000, vol. 1831, pp. 236–254.

[15] Q. Yang and M. Li, "A cut-off approach for bounded verification of parameterized systems," in *ICSE '10*. ACM, 2010, pp. 345–354.

[16] Y. Hanna, D. Samuelson, S. Basu, and H. Rajan, "Automating cut-off for multi-parameterized systems," in *ICFEM '10*, ser. LNCS. Springer, 2010, vol. 6447, pp. 338–354.

[17] E. A. Emerson and V. Kahlon, "Model checking large-scale and parameterized resource allocation systems," in *TACAS '02*, ser. LNCS. Springer, 2002, vol. 2280, pp. 251–265.

[18] ——, "Exact and efficient verification of parameterized cache coherence protocols," in *CHARME '03*, ser. LNCS, vol. 2860. Springer, 2003, pp. 247–262.

[19] E. A. Emerson and K. S. Namjoshi, "On reasoning about rings," *Int. J. Found. Comput. Sci.*, vol. 14, no. 4, pp. 527–550, 2003.

[20] E. A. Emerson and V. Kahlon, "Parameterized model checking of ring-based message passing systems," in *CSL '04*, ser. LNCS, vol. 3210. Springer, 2004, pp. 325–339.

[21] A. Bouajjani, P. Habermehl, and T. Vojnar, "Verification of parametric concurrent systems with prioritised FIFO resource management," *Form. Method. Syst. Des.*, vol. 32, no. 2, pp. 129–172, 2008.

[22] A. Kaiser, D. Kroening, and T. Wahl, "Dynamic cutoff detection in parameterized concurrent programs," in *CAV '10*, ser. LNCS, T. Touili, B. Cook, and P. Jackson, Eds. Springer, 2010, vol. 6174, pp. 645–659.

[23] G. Delzanno, J.-F. Raskin, and L. V. Begin, "Towards the automated verification of multithreaded Java programs," in *TACAS '02*, ser. LNCS. Springer, 2002, vol. 2280, pp. 173–187.

[24] A. Siirtola, "Online appendix," 2013. [Online]. Available: http://users.ics.aalto.fi/sipe/papers/acsd13app.pdf

[25] A. W. Roscoe, *The Theory and Practice of Concurrency*. Prentice Hall, 1997.

[26] A. Siirtola, "Bounds: from parameterised to finite-state verification," in *ACSD '11*. IEEE, 2011, pp. 31–35.

[27] L. De Alfaro and T. Henzinger, "Interface automata," *ACM SIGSOFT Software Engineering Notes*, vol. 26, no. 5, pp. 109–120, 2001.

APPENDIX

A. Additional Definitions, Lemmas and Propositions

For the proofs, we need the following auxiliary concepts and results. For any process P, we write $\mathrm{st}(P)$ for the set of the states of P and $g(P)$ for a process obtained from P by mapping the constants using a function $g : \mathbb{V} \mapsto \mathbb{V}$. For any valuation ϕ, we write $\mathrm{Im}(\phi)$ for the set $\bigcup_{T \in \mathrm{dom}(\phi) \cap \mathbb{T}} \phi(T)$ of all constants occurring in the image of ϕ. Notation $g \circ \psi$, where ψ is a valuation and $g : \mathbb{V} \mapsto \mathbb{V}$ a bijection, denotes a function ϕ with the domain $\mathrm{dom}(\psi)$ such that $\phi(x) = g(\psi(x))$ for all variables $x \in \mathrm{dom}(\psi)$ and $\phi(T) = \{g(v) \mid v \in \psi(T)\}$ for all types $T \in \mathrm{dom}(\psi)$. Hence, valuations ϕ_1 and ϕ_2 are isomorphic if there is a bijection $g : \mathbb{V} \mapsto \mathbb{V}$ such that $\phi_2 = g \circ \phi_1$.

Proposition 23. *Let P_1, P_2 and P_3 be processes, E a set of visible channels and g a bijection: $\mathbb{V} \mapsto \mathbb{V}$. Then the following holds.*

1) $P_1 \parallel P_2 \equiv_{\mathrm{tr}} P_2 \parallel P_1$,
2) $P_1 \parallel (P_2 \parallel P_3) \equiv_{\mathrm{tr}} (P_1 \parallel P_2) \parallel P_3$,
3) $P_1 \parallel P_1 \equiv_{\mathrm{tr}} P_1$,
4) *if* $\mathrm{alph}(P_2) = \emptyset$, *then* $P_1 \parallel P_2 \equiv_{\mathrm{tr}} P_1$,
5) P_1 *is deterministic if and only if* $g(P_1)$ *is deterministic,*
6) $P_1 \parallel P_2$ *is deterministic if and only if both* P_1 *and* P_2 *are deterministic.*
7) $g(P_1 \parallel P_2) = g(P_1) \parallel g(P_2)$,
8) $g(P_1 \setminus E) = g(P_1) \setminus E$,
9) $(P_1 \parallel P_2) \setminus E \preceq_{\mathrm{tr}} (P_1 \setminus E) \parallel (P_2 \setminus E)$, *and*
10) *if* $P_1 \preceq_{\mathrm{tr}} P_2$, *then* $P_1 \parallel P_3 \preceq_{\mathrm{tr}} P_2 \parallel P_3$, $g(P_1) \preceq_{\mathrm{tr}} g(P_2)$ *and* $P_1 \setminus E \preceq_{\mathrm{tr}} P_2 \setminus E$.

Proposition 24. *Let $\mathcal{P}$ be a CPP and ϕ a compatible valuation. Then (i) $[\![\mathcal{P}]\!]_\phi$ is a finite process and (ii) $[\![\mathcal{P}]\!]_\phi = [\![\mathcal{P}]\!]_\psi$ whenever ψ is a valuation such that $\psi|_{\mathrm{par}(\mathcal{P})} = \phi|_{\mathrm{par}(\mathcal{P})}$.*

Lemma 25. *Let $\mathcal{C}$ be a guard, ψ a compatible valuation and ϕ a subvaluation of ψ. Then $[\![\mathcal{C}]\!]_\psi$ is $true$ if and only if $[\![\mathcal{C}]\!]_\phi$ is $true$.*

Lemma 26. *Let $\mathcal{S}$ be an SPP, ψ a compatible valuation and ϕ a $\mathbb{T}_P$-subvaluation of ψ. Then $[\![\mathcal{S}]\!]_\psi = [\![\mathcal{S}]\!]_\phi$.*

Lemma 27. *Let $\mathcal{C}$ be a guard, ϕ a compatible valuation and g a bijection: $\mathbb{V} \mapsto \mathbb{V}$ such that $g(I_T) = I_T$ for every type T. Then $[\![\mathcal{C}]\!]_\phi$ if and only if $[\![\mathcal{C}]\!]_{g \circ \phi}$.*

Lemma 28. *Let $\mathcal{R}$ be a CPP, ϕ a compatible valuation and g a bijection: $\mathbb{V} \mapsto \mathbb{V}$ such that $g(I_T) = I_T$ for all types T. Then $g([\![\mathcal{R}]\!]_\phi) = [\![\mathcal{R}]\!]_{g \circ \phi}$.*

Proposition 29. *If $\mathcal{R}$ is a CPP, ϕ a compatible valuation, T a data type and g a bijection on $\mathbb{V}$ which moves only constants in $\phi(T) \setminus \mathrm{im}(\phi|_{\mathbb{X}})$, then $[\![\mathcal{R}]\!]_\phi = g([\![\mathcal{R}]\!]_\phi)$.*

Lemma 30. *Let $\mathcal{R}$ be a CPP, ψ a compatible valuation, $T \in \mathrm{dom}(\psi)$ a data type and ϕ a $\{T\}$-subvaluation of ψ such that $|\phi(T)| \geq \mathrm{idc}\big([\![\mathcal{R}]\!]_\psi, I_T \setminus \mathrm{im}(\psi|_{\mathbb{X}})\big) + |\mathrm{im}(\psi|_{\mathbb{X}}) \cap I_T|$. Then (i) the initial state of $[\![\mathcal{R}]\!]_\phi$ is the initial state of $[\![\mathcal{R}]\!]_\psi$ and (ii) whenever $\mathbf{d}, \mathbf{d}'$ and $\mathbf{e}$ are tuples of constants over $\mathrm{Im}(\phi)$, then $(s\,\mathbf{d}, c\,\mathbf{e}, s'\mathbf{d}') \in \mathrm{tran}([\![\mathcal{R}]\!]_\phi)$ if and only if $(s\,\mathbf{d}, c\,\mathbf{e}, s'\mathbf{d}') \in \mathrm{tran}([\![\mathcal{R}]\!]_\psi)$.*

Lemma 31. *Let Φ be a set of non-isomorphic valuations with the same finite domain and k a positive integer such that $|\phi(T)| \leq k$ for all valuations $\phi \in \Phi$ and types $T \in \mathrm{dom}(\phi)$. Then the set Φ is finite.*

Proposition 32. *Let $\mathcal{P}$ and $\mathcal{Q}$ be CPPs and ϕ and ψ isomorphic valuations compatible with the CPPs. If $[\![\mathcal{P}]\!]_\phi \preceq_{\mathrm{tr}} [\![\mathcal{Q}]\!]_\phi$ then $[\![\mathcal{P}]\!]_\psi \preceq_{\mathrm{tr}} [\![\mathcal{Q}]\!]_\psi$, too.*

B. Proofs

Proposition 23: Most of the properties are well known [2], [3]. See [5] for proofs. ∎

Proposition 24: The proposition follows straightforwardly from the definition of the instance. ∎

Proof: The relation $\widehat{\preceq}_{\mathrm{tr}}$ is obviously reflexive because $[\![\mathcal{P}]\!]_\phi \preceq_{\mathrm{tr}} [\![\mathcal{P}]\!]_\phi$ for all CPPs $\mathcal{P}$ and valuations ϕ such that $\mathrm{par}(\mathcal{P}) \subseteq \mathrm{dom}(\phi)$.

To see that $\widehat{\preceq}_{\mathrm{tr}}$ is also transitive, let $\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3$ be CPPs such that $\mathcal{P}_1 \mathrel{\widehat{\preceq}_{\mathrm{tr}}} \mathcal{P}_2$ and $\mathcal{P}_2 \mathrel{\widehat{\preceq}_{\mathrm{tr}}} \mathcal{P}_3$. Moreover, let ϕ be a valuation compatible with $\mathcal{P}_1$ and $\mathcal{P}_3$ and ϕ' any valuation with the domain $\mathrm{dom}(\phi) \cup \mathrm{par}(\mathcal{P}_2)$ such that $\phi'|_{\mathrm{dom}(\phi)} = \phi$. Note that such a valuation always exists. By assumption, $[\![\mathcal{P}_1]\!]_{\phi'} \preceq_{\mathrm{tr}} [\![\mathcal{P}_2]\!]_{\phi'}$ and $[\![\mathcal{P}_2]\!]_{\phi'} \preceq_{\mathrm{tr}} [\![\mathcal{P}_3]\!]_{\phi'}$, which by the transitivity of $\preceq_{\mathrm{tr}}$ implies that $[\![\mathcal{P}_1]\!]_{\phi'} \preceq_{\mathrm{tr}} [\![\mathcal{P}_3]\!]_{\phi'}$. By Proposition 24, it means that $[\![\mathcal{P}_1]\!]_\phi \preceq_{\mathrm{tr}} [\![\mathcal{P}_3]\!]_\phi$, too. Therefore, $\mathcal{P}_1 \mathrel{\widehat{\preceq}_{\mathrm{tr}}} \mathcal{P}_3$, which implies that $\widehat{\preceq}_{\mathrm{tr}}$ is transitive.

Finally, we need to show that $\widehat{\preceq}_{\mathrm{tr}}$ is compositional. For that purpose, let $\mathcal{P}_1$ and $\mathcal{P}_2$ be CPPs such that $\mathcal{P}_1 \mathrel{\widehat{\preceq}_{\mathrm{tr}}} \mathcal{P}_2$.

If $\mathcal{C}$ is a guard and ϕ a valuation compatible with $\mathcal{C}, \mathcal{P}_1$ and $\mathcal{P}_2$, then there are two cases to consider. If $[\![\mathcal{C}]\!]_\phi$ is $false$, then obviously $[\![[\mathcal{C}]\mathcal{P}_1]\!]_\phi \preceq_{\mathrm{tr}} [\![[\mathcal{C}]\mathcal{P}_2]\!]_\phi$. On the other hand, if $[\![\mathcal{C}]\!]_\phi$ is $true$, then $[\![[\mathcal{C}]\mathcal{P}_1]\!]_\phi = [\![\mathcal{P}_1]\!]_\phi \preceq_{\mathrm{tr}} [\![\mathcal{P}_2]\!]_\phi = [\![[\mathcal{C}]\mathcal{P}_2]\!]_\phi$. Hence, $\mathcal{P}_1 \mathrel{\widehat{\preceq}_{\mathrm{tr}}} \mathcal{P}_2$ implies that $[\mathcal{C}]\mathcal{P}_1 \mathrel{\widehat{\preceq}_{\mathrm{tr}}} [\mathcal{C}]\mathcal{P}_2$, too.

Similarly, we can show that $\mathcal{P}_1 \mathrel{\widehat{\preceq}_{\mathrm{tr}}} \mathcal{P}_2$ implies $\mathcal{P} \mathbin{\widehat{\parallel}} \mathcal{P}_1 \mathrel{\widehat{\preceq}_{\mathrm{tr}}} \mathcal{P} \mathbin{\widehat{\parallel}} \mathcal{P}_2$, $\mathcal{P}_1 \mathbin{\widehat{\parallel}} \mathcal{P} \mathrel{\widehat{\preceq}_{\mathrm{tr}}} \mathcal{P}_2 \mathbin{\widehat{\parallel}} \mathcal{P}$, $\mathcal{P}_1 \mathbin{\widehat{\setminus}} E \mathrel{\widehat{\preceq}_{\mathrm{tr}}} \mathcal{P}_2 \mathbin{\widehat{\setminus}} E$ and $\mathbin{\widehat{\parallel}_x} \mathcal{P}_1 \mathrel{\widehat{\preceq}_{\mathrm{tr}}} \mathbin{\widehat{\parallel}_x} \mathcal{P}_2$ for all CPPs $\mathcal{P}$, all process variables x and all finite sets E of visible channels. Therefore, the relation $\widehat{\preceq}_{\mathrm{tr}}$ is also compositional and hence a precongruence. ∎

Lemma 25: The lemma follows easily by induction on the structure of $\mathcal{C}$ by using the lemma as an induction hypothesis. ∎

Lemma 26: Since process variables can occur only free in $\mathcal{S}$, it is obvious that both $[\![\mathcal{S}]\!]_\psi$ and $[\![\mathcal{S}]\!]_\phi$ have the same initial state and the same set of transitions. ∎

Lemma 13: First, we show that when $\mathcal{R}$ does not involve hiding, $[\![\mathcal{R}]\!]_\psi \equiv_{\mathrm{tr}} \|_{\phi \in \Phi} [\![\mathcal{R}]\!]_\phi$. We argue by induction on the structure of $\mathcal{R}$ by using the claim as an induction hypothesis.

In the base case, $\mathcal{R}$ is an SPP. Since $\mathrm{prc}(\mathcal{R}, \psi) = \bigcup_{\phi \in \Phi} \mathrm{prc}(\mathcal{R}, \phi)$, it means that $[\![\mathcal{R}]\!]_\psi = [\![\mathcal{R}]\!]_\phi$ for all $\phi \in \Phi$.

Since every process is an idempotent with respect to the parallel composition (Prop. 23, Item 3), it implies that $[\![\mathcal{R}]\!]_\psi \equiv_{\rm tr} \|_{\phi\in\Phi}[\![\mathcal{R}]\!]_\phi$.

In the induction step, there are three cases to consider. First, let us assume that $\mathcal{R}$ is $[\mathcal{C}]\mathcal{R}'$. Since ϕ is a subvaluation of ψ for all $\phi \in \Phi$, by Lemma 25, $[\![\mathcal{C}]\!]_\psi = [\![\mathcal{C}]\!]_\phi$ for every $\phi \in \Phi$. If $[\![\mathcal{C}]\!]_\psi$ is not true, then $[\![\mathcal{R}]\!]_\psi = [\![\mathcal{R}]\!]_\phi = P_{id}$ for all $\phi \in \Phi$. By the idempotence of the parallel composition (Prop. 23, Item 3), it implies that $[\![\mathcal{R}]\!]_\psi \equiv_{\rm tr} \|_{\phi\in\Phi}[\![\mathcal{R}]\!]_\phi$. Otherwise, if $[\![\mathcal{C}]\!]_\psi$ is true, $[\![\mathcal{R}]\!]_\psi = [\![\mathcal{R}']\!]_\psi$ and $[\![\mathcal{R}]\!]_\phi = [\![\mathcal{R}']\!]_\phi$ for all $\phi \in \Phi$. It also means that

$$\begin{aligned}\mathrm{prc}(\mathcal{R}',\psi) &= \mathrm{prc}(\mathcal{R},\psi)\\ &= \bigcup_{\phi\in\Phi}\mathrm{prc}(\mathcal{R},\phi) = \bigcup_{\phi\in\Phi}\mathrm{prc}(\mathcal{R}',\phi)\ .\end{aligned}$$

Hence, by the induction hypothesis,

$$[\![\mathcal{R}]\!]_\psi = [\![\mathcal{R}']\!]_\psi \equiv_{\rm tr} \|_{\phi\in\Phi}[\![\mathcal{R}']\!]_\phi = \|_{\phi\in\Phi}[\![\mathcal{R}]\!]_\phi\ .$$

Next, we assume that $\mathcal{R}$ is $\mathcal{R}_1 \mathbin{\widehat{\|}} \mathcal{R}_2$. Since $\mathrm{prc}(\mathcal{R},\psi) = \bigcup_{\phi\in\Phi}\mathrm{prc}(\mathcal{R},\phi)$, it implies that $\mathrm{prc}(\mathcal{R}_i,\psi) = \bigcup_{\phi\in\Phi}\mathrm{prc}(\mathcal{R}_i,\phi)$ for both $i \in \{1,2\}$. Then, by the induction hypothesis, $[\![\mathcal{R}_i]\!]_\psi \equiv_{\rm tr} \|_{\phi\in\Phi}[\![\mathcal{R}_i]\!]_\phi$ for both $i \in \{1,2\}$. By the commutativity and associativity of the parallel composition (Prop. 23, Items 1–2) and the compositionality of the trace equivalence (Prop. 23, Item 10),

$$\begin{aligned}[\![\mathcal{R}]\!]_\psi &= [\![\mathcal{R}_1]\!]_\psi \parallel [\![\mathcal{R}_2]\!]_\psi\\ &\overset{\text{i.h. \& P23.10}}{\equiv_{\rm tr}} (\|_{\phi\in\Phi}[\![\mathcal{R}_1]\!]_\phi) \parallel (\|_{\phi\in\Phi}[\![\mathcal{R}_2]\!]_\phi)\\ &\overset{\text{P23.1–2,10}}{\equiv_{\rm tr}} \|_{\phi\in\Phi}([\![\mathcal{R}_1]\!]_\phi \parallel [\![\mathcal{R}_2]\!]_\phi) = \|_{\phi\in\Phi}[\![\mathcal{R}]\!]\ .\end{aligned}$$

By the transitivity of the trace equivalence, it implies that $[\![\mathcal{R}]\!]_\psi \equiv_{\rm tr} \|_{\phi\in\Phi}[\![\mathcal{R}]\!]_\phi$.

Finally, let $\mathcal{R}$ be $\widehat{\|}_x \mathcal{R}'$. For every $\psi' \in \mathrm{ext}(\psi,\{x\})$, let $\Phi_{\psi'}$ denote the set of all $\phi' \in \bigcup_{\phi\in\Phi}\mathrm{ext}(\phi,\{x\})$ such that $\phi'(x) = \psi'(x)$. Obviously, $\Phi_{\psi'}$ is a finite set of $\{T\}$-subvaluations of ψ' for every $\psi' \in \mathrm{ext}(\psi,\{x\})$. Moreover, since $\mathrm{prc}(\mathcal{R},\psi) = \bigcup_{\phi\in\Phi}\mathrm{prc}(\mathcal{R},\phi)$, then $\mathrm{prc}(\mathcal{R},\psi')$ must be $\bigcup_{\phi'\in\Phi_{\psi'}}\mathrm{prc}(\mathcal{R},\phi')$ for all $\psi' \in \mathrm{ext}(\psi,\{x\})$, too. By the induction hypothesis, it implies that $[\![\mathcal{R}']\!]_{\psi'} \equiv_{\rm tr} \|_{\phi'\in\Phi_{\psi'}}[\![\mathcal{R}']\!]_{\phi'}$. Then, by the commutativity, associativity and idempotence of the parallel composition (Prop. 23, Items 1–3), the identity of P_{id} (Prop. 23, Item 4) and the compositionality of the trace equivalence (Prop. 23, Item 10),

$$\begin{aligned}[\![\mathcal{R}]\!]_\psi &= \|_{\psi'\in\mathrm{ext}(\psi,\{x\})}[\![\mathcal{R}]\!]_{\psi'}\\ &\overset{\text{i.h. \& P23.4,10}}{\equiv_{\rm tr}} \|_{\psi'\in\mathrm{ext}(\psi,\{x\})}(\|_{\phi'\in\Phi_{\psi'}}[\![\mathcal{R}']\!]_{\phi'})\\ &\overset{\text{P23.1–2,4}}{\equiv_{\rm tr}} \|_{\phi'\in\bigcup_{\phi\in\Phi}\mathrm{ext}(\phi,\{x\})}[\![\mathcal{R}']\!]_{\phi'}\\ &\overset{\text{P23.1–4}}{\equiv_{\rm tr}} \|_{\phi\in\Phi}(\|_{\phi'\in\mathrm{ext}(\phi,\{x\})}[\![\mathcal{R}']\!]_{\phi'}) = \|_{\phi\in\Phi}[\![\mathcal{R}]\!]_\phi\ .\end{aligned}$$

By the transitivity of the trace equivalence, it implies that $[\![\mathcal{R}]\!]_\psi \equiv_{\rm tr} \|_{\phi\in\Phi}[\![\mathcal{R}]\!]_\phi$. Hence, by the induction principle, the first part of the lemma is correct.

Next, we prove that if $\mathcal{R}$ is allowed to involve hiding, $[\![\mathcal{R}]\!]_\psi \preceq_{\rm tr} \|_{\phi\in\Phi}[\![\mathcal{R}]\!]_\phi$. The proof is similar to the above one except for a case, where $\mathcal{R}$ is $\mathcal{R}'\widehat{\setminus}E$. To see that the claim holds in this case, too, we apply the induction hypothesis, the distributivity of hiding (Prop. 23, Item 9) and the compositionality of the trace refinement (Prop. 23, Item 10), which gives us

$$\begin{aligned}[\![\mathcal{R}]\!]_\psi = [\![\mathcal{R}']\!]_\psi \setminus E &\overset{\text{i.h. \& P23.10}}{\preceq_{\rm tr}} (\|_{\phi\in\Phi}[\![\mathcal{R}']\!]_\phi) \setminus E\\ &\overset{\text{P23.9–10}}{\preceq_{\rm tr}} \|_{\phi\in\Phi}([\![\mathcal{R}']\!]_\phi \setminus E) = \|_{\phi\in\Phi}[\![\mathcal{R}]\!]_\phi\ .\end{aligned}$$

By the transitivity of the trace refinement, it implies that $[\![\mathcal{R}]\!]_\psi \preceq_{\rm tr} \|_{\phi\in\Phi}[\![\mathcal{R}]\!]_\phi$ and the lemma is correct. ■

Proposition 14: Let $[\![\mathcal{P}]\!]_\phi \preceq_{\rm tr} [\![\mathcal{Q}]\!]_\phi$ for all valuations $\phi \in \Phi$. Then, by the compositionality of the trace refinement (Prop. 23, Item 10), $\|_{\phi\in\Phi}[\![\mathcal{P}]\!]_\phi \preceq_{\rm tr} \|_{\phi\in\Phi}[\![\mathcal{Q}]\!]_\phi$. By Lemma 13, it implies that $[\![\mathcal{P}]\!]_\psi \preceq_{\rm tr} \|_{\phi\in\Phi}[\![\mathcal{P}]\!]_\phi$ and $[\![\mathcal{Q}]\!]_\psi \equiv_{\rm tr} \|_{\phi\in\Phi}[\![\mathcal{Q}]\!]_\phi$. Hence,

$$[\![\mathcal{P}]\!]_\psi \preceq_{\rm tr} \|_{\phi\in\Phi}[\![\mathcal{P}]\!]_\phi \preceq_{\rm tr} \|_{\phi\in\Phi}[\![\mathcal{Q}]\!]_\phi \equiv_{\rm tr} [\![\mathcal{Q}]\!]_\psi\ ,$$

which by the transitivity of the trace refinement implies that $[\![\mathcal{P}]\!]_\psi \preceq_{\rm tr} [\![\mathcal{Q}]\!]_\psi$. ■

Lemma 15: We argue by induction on the structure of $\mathcal{R}$ by using the lemma as an induction hypothesis.

In the base step, $\mathcal{R}$ is an SPP $\mathcal{S}$. First, note that Φ is non-empty because $\min(k,|\psi(T)|) \geq |\mathrm{im}(\psi|_{\mathbb{X}}) \cap I_T|$. Secondly, since each $\phi \in \Phi$ is a $\{T\}$-subvaluation of ψ, by Lemma 26, it means that $[\![\mathcal{S}]\!]_\psi = [\![\mathcal{S}]\!]_\phi$ for every $\phi \in \Phi$. Then, by definition, it is evident that $\mathrm{prc}(\mathcal{R},\psi) = \bigcup_{\phi\in\Phi}\mathrm{prc}(\mathcal{R},\phi)$.

In the induction step, there are four cases to consider. First, let us assume that $\mathcal{R}$ is $[\mathcal{C}]\mathcal{R}'$. Since ϕ is a subvaluation of ψ for all $\phi \in \Phi$, by Lemma 25, $[\![\mathcal{C}]\!]_\psi = [\![\mathcal{C}]\!]_\phi$ for every $\phi \in \Phi$. If $[\![\mathcal{C}]\!]_\psi$ is *false*, then $\mathrm{prc}(\mathcal{R},\psi) = \bigcup_{\phi\in\Phi}\mathrm{prc}(\mathcal{R},\phi) = \emptyset$. Otherwise, if $[\![\mathcal{C}]\!]_\psi$ is *true*, then by definition and the induction hypothesis,

$$\begin{aligned}\mathrm{prc}(\mathcal{R},\psi) &= \mathrm{prc}(\mathcal{R}',\psi)\\ &= \bigcup_{\phi\in\Phi}\mathrm{prc}(\mathcal{R}',\phi) = \bigcup_{\phi\in\Phi}\mathrm{prc}(\mathcal{R},\phi)\ .\end{aligned}$$

Next, let $\mathcal{R}$ be $\mathcal{R}_1 \mathbin{\widehat{\|}} \mathcal{R}_2$. Obviously, $k \geq |\mathrm{im}(\psi|_{\mathbb{X}}) \cap I_T| + \mathrm{pbnd}_T(\mathcal{R}) \geq |\mathrm{im}(\psi|_{\mathbb{X}}) \cap I_T| + \mathrm{pbnd}_T(\mathcal{R}_i)$ for both $i \in$

$\{1,2\}$, which means that the induction hypothesis is applicable to $\mathcal{R}_1$ and $\mathcal{R}_2$. Hence, $\mathrm{prc}(\mathcal{R}_i,\psi) = \bigcup_{\phi\in\Phi}\mathrm{prc}(\mathcal{R}_i,\phi)$ for both $i\in\{1,2\}$, which implies that

$$\begin{aligned}\mathrm{prc}(\mathcal{R},\psi) &= \bigcup_{i\in\{1,2\}}\big(\{i\}\times\mathrm{prc}(\mathcal{R}_i,\psi)\big)\\ &= \bigcup_{i\in\{1,2\}}\big(\{i\}\times\bigcup_{\phi\in\Phi}\mathrm{prc}(\mathcal{R}_i,\phi)\big)\\ &= \bigcup_{\phi\in\Phi}\bigcup_{i\in\{1,2\}}\big(\{i\}\times\mathrm{prc}(\mathcal{R}_i,\phi)\big) = \bigcup_{\phi\in\Phi}\mathrm{prc}(\mathcal{R},\phi)\ .\end{aligned}$$

After that, we assume that $\mathcal{R}$ is $\mathcal{R}'\mathbin{\widehat{\setminus}}E$. Again, it is easy to see that $k \geq |\mathrm{im}(\psi|_{\mathbb{X}})\cap I_T| + \mathrm{pbnd}_T(\mathcal{R}) \geq |\mathrm{im}(\psi|_{\mathbb{X}})\cap I_T| + \mathrm{pbnd}_T(\mathcal{R}')$, which by definition and the induction hypothesis implies that

$$\begin{aligned}\mathrm{prc}(\mathcal{R},\psi) &= \mathrm{prc}(\mathcal{R}',\psi)\\ &= \bigcup_{\phi\in\Phi}\mathrm{prc}(\mathcal{R}',\phi) = \bigcup_{\phi\in\Phi}\mathrm{prc}(\mathcal{R},\phi)\ .\end{aligned}$$

Finally, we assume that $\mathcal{R}$ is $\widehat{\|}_x\mathcal{R}'$. First, note that whenever $\psi'\in\mathrm{ext}(\psi,\{x\})$, $k \geq |\mathrm{im}(\psi|_{\mathbb{X}})\cap I_T| + \mathrm{pbnd}_T(\mathcal{R}) \geq |\mathrm{im}(\psi'|_{\mathbb{X}})\cap I_T| + \mathrm{pbnd}_T(\mathcal{R}')$. Secondly, for every $\psi'\in\mathrm{ext}(\psi,\{x\})$, let $\Phi_{\psi'}$ be the set of all valuations $\phi'\in\bigcup_{\phi\in\Phi}\mathrm{ext}(\phi,\{x\})$ such that $\phi'(x)=\psi'(x)$. Obviously, $\Phi_{\psi'}$ is a set of $\{T\}$-subvaluations ϕ' of ψ' such that $|\phi'(T)| = \min(k,|\psi(T)|) = \min(k,|\psi'(T)|)$. To see that $\Phi_{\psi'}$ is the set of all such valuations, let ϕ'' be a $\{T\}$-subvaluation of ψ' such that $|\phi''(T)| = \min(k,|\psi'(T)|)$. Then, there is a $\{T\}$-subvaluation ϕ of ψ such that $\phi''\in\mathrm{ext}(\phi,\{x\})$. Since $|\phi(T)| = |\phi''(T)| = \min(k,|\psi'(T)|) = \min(k,|\psi(T)|)$, we know that $\phi\in\Phi$. It implies that $\Phi_{\psi'}$ is the set of all $\{T\}$-subvaluations ϕ' of ψ' such that $|\phi'(T)| = \min(k,|\psi'(T)|)$. By the induction hypothesis, it means that $\mathrm{prc}(\mathcal{R}',\psi') = \bigcup_{\phi'\in\Phi_{\psi'}}\mathrm{prc}(\mathcal{R}',\phi')$ for all $\psi'\in\mathrm{ext}(\psi,\{x\})$. Therefore,

$$\begin{aligned}\mathrm{prc}(\mathcal{R},\psi) &= \bigcup_{\psi'\in\mathrm{ext}(\psi,\{x\})}\big(\{\psi'(x)\}\times\mathrm{prc}(\mathcal{R}',\psi')\big)\\ &= \bigcup_{\psi'\in\mathrm{ext}(\psi,\{x\})}\big(\{\psi'(x)\}\times\bigcup_{\phi'\in\Phi_{\psi'}}\mathrm{prc}(\mathcal{R}',\phi')\big)\\ &= \bigcup_{\psi'\in\mathrm{ext}(\psi,\{x\})}\bigcup_{\phi'\in\Phi_{\psi'}}\big(\{\phi'(x)\}\times\mathrm{prc}(\mathcal{R}',\phi')\big)\\ &= \bigcup_{\phi'\in\bigcup_{\phi\in\Phi}\mathrm{ext}(\phi,\{x\})}\big(\{\phi'(x)\}\times\mathrm{prc}(\mathcal{R}',\phi')\big)\\ &= \bigcup_{\phi\in\Phi}\bigcup_{\phi'\in\mathrm{ext}(\phi,\{x\})}\big(\{\phi'(x)\}\times\mathrm{prc}(\mathcal{R}',\phi')\big)\\ &= \bigcup_{\phi\in\Phi}\mathrm{prc}(\mathcal{R},\phi).\end{aligned}$$

Hence, by the induction principle, the lemma is correct. ■

Theorem 16: Obviously, if $[\![\mathcal{P}]\!]_\phi \preceq_{\mathrm{tr}} [\![\mathcal{Q}]\!]_\phi$ for all $\phi\in\Phi$, then $[\![\mathcal{P}]\!]_\psi \preceq_{\mathrm{tr}} [\![\mathcal{Q}]\!]_\psi$ for all valuations $\psi\in\Psi$, too.

Next, let us assume that $[\![\mathcal{P}]\!]_\psi \preceq_{\mathrm{tr}} [\![\mathcal{Q}]\!]_\psi$ for all valuations $\psi\in\Psi$. Let $T_1,\ldots,T_n$ be the process types in $\overline{\mathrm{par}}(\mathcal{P}\mathbin{\widehat{\|}}\mathcal{Q})$ and $\Psi_0,\ldots,\Psi_n$ sets of valuations with the domain $\overline{\mathrm{par}}(\mathcal{P}\mathbin{\widehat{\|}}\mathcal{Q})$ such that $\Psi_0\subseteq\Psi$, $\Psi_n=\Phi$ and for all $i\in\{1,\ldots,n\}$, Ψ_{i-1} is the set of all $\{T_i\}$-subvaluations ψ_{i-1} of the valuations in Ψ_i such that $|\psi_{i-1}(T_i)| \leq \mathrm{cp}_{T_i}(\mathcal{P}\mathbin{\widehat{\|}}\mathcal{Q})$. We proceed by induction to show that for all $i\in\{0,\ldots,n\}$, the relation $[\![\mathcal{P}]\!]_{\psi_i} \preceq_{\mathrm{tr}} [\![\mathcal{Q}]\!]_{\psi_i}$ holds whenever $\psi_i\in\Psi_i$.

The base step is clear since $\Psi_0\subseteq\Psi$ and the claim holds by assumption. In the induction step, let $\psi_i\in\Psi_i\setminus\Psi_{i-1}$, $k := \mathrm{cp}_{T_i}(\mathcal{P}\mathbin{\widehat{\|}}\mathcal{Q})$ and Φ_i be the set of all $\{T_i\}$-subvaluations ϕ of ψ_i such that $|\phi(T_i)|=k$. By definition, $k \geq |\mathrm{free}_{T_i}(\mathcal{P}\mathbin{\widehat{\|}}\mathcal{Q})| + \mathrm{pbnd}_{T_i}(\mathcal{P}\mathbin{\widehat{\|}}\mathcal{Q}) \geq |\mathrm{im}(\psi_i|_{\mathbb{X}})\cap I_{T_i}| + \max(\mathrm{pbnd}_{T_i}(\mathcal{P}),\mathrm{pbnd}_{T_i}(\mathcal{Q}))$ and by the choice of ψ_i, $|\phi(T_i)| = k = \min(k,|\psi_i(T_i)|)$ for all $\phi\in\Phi_i$. By Lemma 15, it implies that $\mathrm{prc}(\mathcal{Q},\psi_i) = \bigcup_{\phi\in\Phi_i}\mathrm{prc}(\mathcal{Q},\phi)$ and $\mathrm{prc}(\mathcal{P},\psi_i) = \bigcup_{\phi\in\Phi_i}\mathrm{prc}(\mathcal{P},\phi)$. Since $\Phi_i\subseteq\Psi_{i-1}$, by the induction hypothesis, $[\![\mathcal{P}]\!]_\phi \preceq_{\mathrm{tr}} [\![\mathcal{Q}]\!]_\phi$ for all valuations $\phi\in\Phi_i$. By Proposition 14, it means that then $[\![\mathcal{P}]\!]_{\psi_i} \preceq_{\mathrm{tr}} [\![\mathcal{Q}]\!]_{\psi_i}$, too. By the induction principle, it implies that for all $i\in\{0,\ldots,n\}$, $[\![\mathcal{P}]\!]_\psi \preceq_{\mathrm{tr}} [\![\mathcal{Q}]\!]_\psi$ whenever $\psi\in\Psi_i$. Hence, the relation $[\![\mathcal{P}]\!]_\phi \preceq_{\mathrm{tr}} [\![\mathcal{Q}]\!]_\phi$ holds especially for all valuations $\phi\in\Phi$. ■

Lemma 27: The claim follows straightforwardly by induction on the structure of the guard using the lemma as an induction hypothesis. ■

Lemma 28: We argue by induction on the structure of $\mathcal{R}$ by using the lemma as an induction hypothesis.

In the base step, $\mathcal{R}$ is an SPP and it is easy to see that $g([\![\mathcal{R}]\!]_\phi) = [\![\mathcal{R}]\!]_{g\circ\phi}$ holds.

In the induction step, there are four cases to consider. First, we assume that $\mathcal{R}$ is $[\mathcal{C}]\mathcal{R}'$. By Lemma 27, we know that $[\![\mathcal{C}]\!]_\phi$ if and only if $[\![\mathcal{C}]\!]_{g\circ\phi}$. If $[\![\mathcal{C}]\!]_\phi = [\![\mathcal{C}]\!]_{g\circ\phi} = \mathit{false}$, then $[\![\mathcal{R}]\!]_\phi = [\![\mathcal{R}]\!]_{g\circ\phi} = P_{id}$ and the lemma holds trivially. On the other hand, if $[\![\mathcal{C}]\!]_\phi = [\![\mathcal{C}]\!]_{g\circ\phi} = \mathit{true}$, then by the induction hypothesis, $g([\![\mathcal{R}]\!]_\phi) = g([\![\mathcal{R}']\!]_\phi) = [\![\mathcal{R}']\!]_{g\circ\phi} = [\![\mathcal{R}]\!]_{g\circ\phi}$.

Next, if $\mathcal{R}$ is $\mathcal{R}_1\mathbin{\widehat{\|}}\mathcal{R}_2$, then by Item 7 of Proposition 23 and the induction hypothesis,

$$\begin{aligned}g([\![\mathcal{R}]\!]_\phi) &= g([\![\mathcal{R}_1]\!]_\phi \parallel [\![\mathcal{R}_2]\!]_\phi)\\ &\overset{\text{P23.7}}{=} g([\![\mathcal{R}_1]\!]_\phi) \parallel g([\![\mathcal{R}_2]\!]_\phi) \overset{\text{i.h.}}{=} [\![\mathcal{R}_1]\!]_{g\circ\phi} \parallel [\![\mathcal{R}_2]\!]_{g\circ\phi}\\ &= [\![\mathcal{R}]\!]_{g\circ\phi}\ .\end{aligned}$$

After that, let $\mathcal{R}$ be $\mathcal{R}'\mathbin{\widehat{\setminus}}E$. Then by the induction hypothesis, $g([\![\mathcal{R}']\!]_\phi) = [\![\mathcal{R}']\!]_{g\circ\phi}$. By Item 8 of Proposition 23, it implies that

$$\begin{aligned}g([\![\mathcal{R}]\!]_\phi) = g([\![\mathcal{R}']\!]_\phi \setminus E) &\overset{\text{P23.8}}{=} g([\![\mathcal{R}']\!]_\phi)\setminus E\\ &\overset{\text{i.h.}}{=} [\![\mathcal{R}']\!]_{g\circ\phi}\setminus E = [\![\mathcal{R}]\!]_{g\circ\phi}\ .\end{aligned}$$

Finally, if $\mathcal{R}$ is $\widehat{\|}_x \mathcal{R}'$, we argue like in the second case of the induction step. By using Item 7 of Proposition 23 and the induction hypothesis, we see that

$$g([\![\mathcal{R}]\!]_\phi) = g(\|_{\phi' \in \mathrm{ext}(\phi,\{x\})} [\![\mathcal{R}']\!]_{\phi'}) \overset{\text{P23.7}}{=} \|_{\phi' \in \mathrm{ext}(\phi,\{x\})} g([\![\mathcal{R}']\!]_{\phi'}) \overset{\text{i.h.}}{=} \|_{\phi' \in \mathrm{ext}(\phi,\{x\})} [\![\mathcal{R}']\!]_{g\circ\phi'} = \|_{\phi' \in \mathrm{ext}(g\circ\phi,\{x\})} [\![\mathcal{R}']\!]_{\phi'} = [\![\mathcal{R}]\!]_{g\circ\phi} .$$

Hence, also the induction step is clear and by the induction principle, the lemma is correct. ■

Proposition 29: Since $g\circ\phi = \phi$, by Lemma 28, $[\![\mathcal{R}]\!]_\phi = [\![\mathcal{R}]\!]_{g\circ\phi} = g([\![\mathcal{R}]\!]_\phi)$. ■

Lemma 30: We argue by induction on the structure of $\mathcal{R}$ by using the lemma as an induction hypothesis.

In the base case, $\mathcal{R}$ is an SPP $(\hat{s}\,\hat{\mathbf{x}}, \Delta)$. Since ψ is compatible with $\mathcal{R}$ and ϕ is a subvaluation of ψ, $\phi|_{\mathrm{im}(\hat{\mathbf{x}})} = \psi|_{\mathrm{im}(\hat{\mathbf{x}})}$. Hence, $\phi \circ \mathbf{x} = \psi \circ \mathbf{x}$, which implies that the initial states of $[\![\mathcal{R}]\!]_\phi$ and $[\![\mathcal{R}]\!]_\psi$ are the same.

Next, let $\mathbf{d}, \mathbf{d}', \mathbf{e} \in (\mathrm{Im}(\phi))^*$. Since ϕ is a subvaluation of ψ, it easy to see that whenever $(s\,\mathbf{d}, c\,\mathbf{e}, s'\mathbf{d}')$ is a transition of $[\![\mathcal{R}]\!]_\phi$, $(s\,\mathbf{d}, c\,\mathbf{e}, s'\mathbf{d}')$ is a transition of $[\![\mathcal{R}]\!]_\psi$, too. To prove the other direction, let $(s\,\mathbf{d}, c\,\mathbf{e}, s'\mathbf{d}')$ be a transition of $[\![\mathcal{R}]\!]_\psi$. By definition, there are $\mathbf{x}, \mathbf{x}' \in \mathbb{X}_D^*$, $\mathbf{y} \in \mathbb{X}^*$, a guard $\mathcal{C}$, a subset X of $\mathrm{im}(\mathbf{y}\mathbf{x}')$ and $\psi' \in \mathrm{ext}(\psi, \mathrm{im}(\mathbf{x}) \cup X)$ such that $\mathbf{d} = \psi' \circ \mathbf{x}$, $\mathbf{e} = \psi' \circ \mathbf{y}$, $\mathbf{d}' = \psi' \circ \mathbf{x}'$, $(s\,\mathbf{x}, X, \mathcal{C}, c\,\mathbf{y}, s'\mathbf{x}') \in \Delta$ and $[\![\mathcal{C}]\!]_{\psi'}$ is true. Since $\mathbf{d}, \mathbf{d}', \mathbf{e} \in (\mathrm{Im}(\phi))^*$ and ϕ is a subvaluation of ψ, we can pick $\phi' \in \mathrm{ext}(\phi, \mathrm{im}(\mathbf{x}) \cup X)$ such that ϕ' and ψ' agree on the values of the variables in $\mathrm{im}(\mathbf{x}\mathbf{y}\mathbf{x}') \cup \mathrm{par}(\mathcal{C})$. It means that $\mathbf{d} = \phi' \circ \mathbf{x}$, $\mathbf{e} = \phi' \circ \mathbf{y}$, $\mathbf{d}' = \phi' \circ \mathbf{x}'$ and $[\![\mathcal{C}]\!]_{\phi'}$ is true, which implies that $(s\,\mathbf{d}, c\,\mathbf{e}, s'\mathbf{d}')$ is a transition of $[\![\mathcal{R}]\!]_\phi$ as well.

In the induction step, there are four cases to consider. First, we assume that $\mathcal{R}$ is $[\mathcal{C}]\mathcal{R}'$. Because ψ is compatible with $\mathcal{R}$ and ϕ is a subvaluation of ψ, by Lemma 25, it implies that $[\![\mathcal{C}]\!]_\phi$ if and only if $[\![\mathcal{C}]\!]_\psi$. Hence, there are two cases to check. If $[\![\mathcal{C}]\!]_\phi = [\![\mathcal{C}]\!]_\psi = \mathit{false}$, then $[\![\mathcal{R}]\!]_\phi = [\![\mathcal{R}]\!]_\psi = P_{id}$ and it is obvious that the lemma holds. On the other hand, if $[\![\mathcal{C}]\!]_\phi = [\![\mathcal{C}]\!]_\psi = \mathit{true}$, then $[\![\mathcal{R}]\!]_\phi = [\![\mathcal{R}']\!]_\phi$ and $[\![\mathcal{R}]\!]_\psi = [\![\mathcal{R}']\!]_\psi$. Now, by the induction hypothesis, it is again obvious that the lemma holds.

Next, we assume that $\mathcal{R}$ is $\mathcal{R}_1 \,\widehat{\|}\, \mathcal{R}_2$. In order to apply the induction hypothesis to both $\mathcal{R}_1$ and $\mathcal{R}_2$, we need to show that $|\phi(T)| \geq \mathrm{idc}([\![\mathcal{R}_i]\!]_\psi, I_T \setminus \mathrm{im}(\psi|_{\mathbb{X}})) + |\mathrm{im}(\psi|_{\mathbb{X}}) \cap I_T|$ for both $i \in \{1,2\}$. For that purpose, let $(s_1\,\mathbf{d}_1, c\,\mathbf{e}, s_1'\,\mathbf{d}_1')$ be a transition of $[\![\mathcal{R}_1]\!]_\psi$. If $c\,\mathbf{e} \in \mathrm{alph}([\![\mathcal{R}_2]\!]_\psi)$, then there are $s_2\,\mathbf{d}_2, s_2'\,\mathbf{d}_2' \in \mathrm{st}([\![\mathcal{R}_2]\!]_\psi)$ such that $(s_2\,\mathbf{d}_2, c\,\mathbf{e}, s_2'\,\mathbf{d}_2')$ is a transition of $[\![\mathcal{R}_2]\!]_\psi$, which implies that $((s_1, s_2, |\mathbf{d}_1|)\mathbf{d}_1\mathbf{d}_2, c\,\mathbf{e}, (s_1', s_2', |\mathbf{d}_1'|)\mathbf{d}_1'\mathbf{d}_2')$ is a transition of $[\![\mathcal{R}]\!]_\psi$. On the other hand, if $c\,\mathbf{e} \notin \mathrm{alph}([\![\mathcal{R}_2]\!]_\psi)$, then $((s_1, \hat{s}_2, |\mathbf{d}_1|)\mathbf{d}_1\hat{\mathbf{d}}_2, c\,\mathbf{e}, (s_1', \hat{s}_2, |\mathbf{d}_1'|)\mathbf{d}_1'\hat{\mathbf{d}}_2)$ is a transition of $[\![\mathcal{R}]\!]_\psi$, where $\hat{s}_2\,\hat{\mathbf{d}}_2$ is the initial state of $[\![\mathcal{R}_2]\!]_\psi$. Since the initial state of $[\![\mathcal{R}]\!]_\psi$ is the combination of the initial states of $[\![\mathcal{R}_1]\!]_\psi$ and $[\![\mathcal{R}_2]\!]_\psi$, this implies that $\mathrm{idc}([\![\mathcal{R}]\!]_\psi, I_T \setminus \mathrm{im}(\psi|_{\mathbb{X}})) \geq \mathrm{idc}([\![\mathcal{R}_1]\!]_\psi, I_T \setminus \mathrm{im}(\psi|_{\mathbb{X}}))$, which means that $|\phi(T)| \geq \mathrm{idc}([\![\mathcal{R}_1]\!]_\psi, I_T \setminus \mathrm{im}(\psi|_{\mathbb{X}})) + |\mathrm{im}(\psi|_{\mathbb{X}}) \cap I_T|$. Similarly, one can establish $|\phi(T)| \geq \mathrm{idc}([\![\mathcal{R}_2]\!]_\psi, I_T \setminus \mathrm{im}(\psi|_{\mathbb{X}})) + |\mathrm{im}(\psi|_{\mathbb{X}}) \cap I_T|$, too. Now, by the induction hypothesis, $\mathrm{init}([\![\mathcal{R}_i]\!]_\psi) = \mathrm{init}([\![\mathcal{R}_i]\!]_\phi)$ for both $i \in \{1,2\}$, which implies that $[\![\mathcal{R}]\!]_\phi$ and $[\![\mathcal{R}]\!]_\psi$ have the same initial state.

Let us then assume that $((s_1, s_2, |\mathbf{d}_1|)\mathbf{d}_1\mathbf{d}_2, c\,\mathbf{e}, (s_1', s_2', |\mathbf{d}_1'|)\mathbf{d}_1'\mathbf{d}_2')$ is a transition of $[\![\mathcal{R}]\!]_\psi$ such that $\mathbf{d}_1, \mathbf{d}_2, \mathbf{d}_1', \mathbf{d}_2', \mathbf{e} \in (\mathrm{Im}(\phi))^*$. If $(s_i\,\mathbf{d}_i, c\,\mathbf{e}, s_i'\,\mathbf{d}_i') \in \mathrm{tran}([\![\mathcal{R}_i]\!]_\psi)$ for both $i \in \{1,2\}$, then by the induction hypothesis, $(s_i\,\mathbf{d}_i, c\,\mathbf{e}, s_i'\,\mathbf{d}_i') \in \mathrm{tran}([\![\mathcal{R}_i]\!]_\phi)$ for both $i \in \{1,2\}$, which implies that $((s_1, s_2, |\mathbf{d}_1|)\mathbf{d}_1\mathbf{d}_2, c\,\mathbf{e}, (s_1', s_2', |\mathbf{d}_1'|)\mathbf{d}_1'\mathbf{d}_2')$ is a transition of $[\![\mathcal{R}]\!]_\phi$, too. Otherwise, there are distinct $i, j \in \{1,2\}$ such that $(s_i\,\mathbf{d}_i, c\,\mathbf{e}, s_i'\,\mathbf{d}_i') \in \mathrm{tran}([\![\mathcal{R}_i]\!]_\psi)$, $c\,\mathbf{e} \notin \mathrm{alph}([\![\mathcal{R}_j]\!]_\psi)$, and $s_j\,\mathbf{d}_j$ and $s_j'\,\mathbf{d}_j'$ are the same state of $[\![\mathcal{R}_j]\!]_\psi$. Then, by the induction hypothesis, it is evident that $(s_i\,\mathbf{d}_i, c\,\mathbf{e}, s_i'\,\mathbf{d}_i') \in \mathrm{tran}([\![\mathcal{R}_i]\!]_\phi)$. To see that $c\,\mathbf{e} \notin \mathrm{alph}([\![\mathcal{R}_j]\!]_\phi)$, assume contrarywise that $c\,\mathbf{e} \in \mathrm{alph}([\![\mathcal{R}_j]\!]_\phi)$. Then, by definition, $c\,\mathbf{e}$ appears in a transition of $[\![\mathcal{R}_j]\!]_\phi$, which by the induction hypothesis, implies that also $[\![\mathcal{R}_j]\!]_\psi$ has the same transition. Therefore, $c\,\mathbf{e} \in \mathrm{alph}([\![\mathcal{R}_j]\!]_\psi)$, which is a contradiction. Hence, $c\,\mathbf{e} \notin \mathrm{alph}([\![\mathcal{R}_j]\!]_\phi)$. Because $s_j\,\mathbf{d}_j \in \mathrm{st}([\![\mathcal{R}_j]\!]_\psi)$, $s_j\,\mathbf{d}_j$ is the initial state or there is a transition $(s_3\,\mathbf{d}_3, c_3\,\mathbf{e}_3, s_3'\,\mathbf{d}_3')$ of $[\![\mathcal{R}_j]\!]_\psi$ such that $s_j\,\mathbf{d}_j \in \{s_3\,\mathbf{d}_3, s_3'\,\mathbf{d}_3'\}$. If $s_j\,\mathbf{d}_j = \mathrm{init}([\![\mathcal{R}_j]\!]_\psi)$, then by the induction hypothesis, $s_j\,\mathbf{d}_j = \mathrm{init}([\![\mathcal{R}_j]\!]_\phi)$. Otherwise, since $|\phi(T)| \geq \mathrm{idc}([\![\mathcal{R}_j]\!]_\psi, I_T \setminus \mathrm{im}(\psi|_{\mathbb{X}})) + |\mathrm{im}(\psi|_{\mathbb{X}}) \cap I_T| \geq |(\mathrm{im}(\psi|_{\mathbb{X}}) \cup \mathrm{im}(\mathbf{d}_3\mathbf{e}_3\mathbf{d}_3')) \cap I_T|$, we can pick a bijection $g : \mathbb{V} \mapsto \mathbb{V}$ such that $\mathrm{im}(g \circ \mathbf{d}_3\mathbf{e}_3\mathbf{d}_3') \subseteq \mathrm{Im}(\phi)$ and g moves only constants in $\psi(T) \setminus (\mathrm{im}(\psi|_{\mathbb{X}}) \cup \mathrm{im}(\mathbf{d}_j))$. Then, by Proposition 29, $(s_3(g \circ \mathbf{d}_3), c_3(g \circ \mathbf{e}_3), s_3'(g \circ \mathbf{d}_3')) \in \mathrm{tran}([\![\mathcal{R}_j]\!]_\psi)$ such that $g \circ \mathbf{d}_3, g \circ \mathbf{e}_3, g \circ \mathbf{d}_3' \in (\mathrm{Im}(\phi))^*$, which by the induction hypothesis implies that $(s_3(g \circ \mathbf{d}_3), c_3(g \circ \mathbf{e}_3), s_3'(g \circ \mathbf{d}_3')) \in \mathrm{tran}([\![\mathcal{R}_j]\!]_\phi)$, too. Therefore, $s_j\,\mathbf{d}_j$ is a state of $[\![\mathcal{R}_j]\!]_\phi$, which implies that also in this case, $((s_1, s_2, |\mathbf{d}_1|)\mathbf{d}_1\mathbf{d}_2, c\,\mathbf{e}, (s_1', s_2', |\mathbf{d}_1'|)\mathbf{d}_1'\mathbf{d}_2')$ is a transition of $[\![\mathcal{R}]\!]_\phi$.

Proving the other direction, i.e., the containment $\mathrm{tran}([\![\mathcal{R}]\!]_\phi) \subseteq \mathrm{tran}([\![\mathcal{R}]\!]_\psi)$, proceeds similarly. The main difference is that now it is easy to see that a state of $[\![\mathcal{R}]\!]_\phi$ is a state of $[\![\mathcal{R}]\!]_\psi$, too, while proving that $c\,\mathbf{e} \notin \mathrm{alph}([\![\mathcal{R}_j]\!]_\phi)$ implies $c\,\mathbf{e} \notin \mathrm{alph}([\![\mathcal{R}_j]\!]_\psi)$ is a similar process as establishing $s_j\,\mathbf{d}_j \in [\![\mathcal{R}_j]\!]_\phi$ above.

Thirdly, we assume that $\mathcal{R}$ is $\mathcal{R}' \,\widehat{\setminus}\, E$. Since hiding affects only channel names, it is evident that $\mathrm{idc}([\![\mathcal{R}]\!]_\phi, I_T \setminus \mathrm{im}(\psi|_{\mathbb{X}})) = \mathrm{idc}([\![\mathcal{R}']\!]_\phi, I_T \setminus \mathrm{im}(\psi|_{\mathbb{X}}))$, which implies that the induction hypothesis is applicable to $\mathcal{R}'$. By definition and induction hypothesis, it is then straightforward to check that $[\![\mathcal{R}]\!]_\phi$ and $[\![\mathcal{R}']\!]_\psi$ have the same initial state.

If $(s\,\mathbf{d}, c\,\mathbf{e}, s'\mathbf{d}')$ is a transition of $[\![\mathcal{R}]\!]_\psi$ such that $\mathbf{d}, \mathbf{d}', \mathbf{e} \in (\mathrm{Im}(\phi))^*$, then there are two cases to consider. If $(s\,\mathbf{d}, c\,\mathbf{e}, s'\mathbf{d}') \in \mathrm{tran}([\![\mathcal{R}']\!]_\psi)$ and $c \notin E$, then the induction hypothesis implies that $(s\,\mathbf{d}, c\,\mathbf{e}, s'\mathbf{d}') \in \mathrm{tran}([\![\mathcal{R}']\!]_\phi)$ as well. Therefore, $(s\,\mathbf{d}, c\,\mathbf{e}, s'\mathbf{d}')$ is a transition of $[\![\mathcal{R}]\!]_\phi$, too. Otherwise, $c = \tau$ and there is $c' \in E$ such that $(s\,\mathbf{d}, c'\mathbf{e}, s'\mathbf{d}') \in \mathrm{tran}([\![\mathcal{R}']\!]_\psi)$. By the induction hypothesis, it implies that $(s\,\mathbf{d}, c'\mathbf{e}, s'\mathbf{d}') \in \mathrm{tran}([\![\mathcal{R}']\!]_\phi)$, which, in turn, means that $(s\,\mathbf{d}, c\,\mathbf{e}, s'\mathbf{d}')$ is a transition of $[\![\mathcal{R}]\!]_\phi$ also in this case. Proving $\mathrm{tran}([\![\mathcal{R}]\!]_\phi) \subseteq \mathrm{tran}([\![\mathcal{R}]\!]_\psi)$, proceeds similarly.

The case when $\mathcal{R}$ is $\|_x \mathcal{R}'$ is a generalisation of the case when $\mathcal{R}$ is $\mathcal{R}_1 \| \mathcal{R}_2$ and can be treated in a similar way. That is because x is a process variable, so for every $\psi' \in \mathrm{ext}(\psi, \{x\})$ there is $\phi' \in \mathrm{ext}(\phi, \{x\})$ such that ϕ' is a subvaluation of ψ'. Hence, also this case is clear and, by the induction principle, the lemma is correct. ∎

Lemma 17: We argue by induction on k by using the lemma as an induction hypothesis.

In the base case, $k = 0$, which implies that the execution $s_0\,\mathbf{d}_0$ is the initial state of $[\![\mathcal{P}]\!]_\psi$. By Lemma 30, $s_0\,\mathbf{d}_0$ is an execution of $[\![\mathcal{P}]\!]_\phi$, too. Moreover, by the same lemma, the initial state $s'_0\,\mathbf{d}'_0$ of $[\![\mathcal{Q}]\!]_\phi$ is an execution of $[\![\mathcal{Q}]\!]_\phi$ such that $s'_0\,\mathbf{d}'_0$ is an execution of $[\![\mathcal{Q}]\!]_\psi$, too. Hence, the base case is clear.

In the induction step, k is a positive integer. Now, we can apply the induction hypothesis to the execution $(s_0\,\mathbf{d}_0, c_1\,\mathbf{e}_1, s_1\,\mathbf{d}_1, \ldots, c_m\,\mathbf{e}_m, s_m\,\mathbf{d}_m)$, where $m = i_{k-1}$. Hence, (i) there are bijections $g_1, \ldots, g_m : \mathbb{V} \mapsto \mathbb{V}$ which move only constants in $\psi(T) \setminus \mathrm{im}(\psi|_{\mathbb{X}})$ such that $(s_0\,\mathbf{d}_0, c_1(g_1 \circ \mathbf{e}_1), s_1(g_1 \circ \mathbf{d}_1), \ldots, c_m(g_m \circ \mathbf{e}_m), s_m(g_m \circ \mathbf{d}_m))$ is an execution of $[\![\mathcal{P}]\!]_\phi$ and (ii) there is an execution $(s''_0\,\mathbf{d}''_0, c''_1\,\mathbf{e}''_1, s''_1\,\mathbf{d}''_1, \ldots, c''_{k-1}\,\mathbf{e}''_{k-1}, s''_{k-1}\,\mathbf{d}''_{k-1})$ of $[\![\mathcal{Q}]\!]_\phi$ such that $(s''_0\,\mathbf{d}''_0, c''_1(g_{i_1}^{-1} \circ \mathbf{e}''_1), s''_1(g_{i_1}^{-1} \circ \mathbf{d}''_1), \ldots, c''_{k-1}(g_m^{-1} \circ \mathbf{e}''_{k-1}), s''_{k-1}(g_m^{-1} \circ \mathbf{d}''_{k-1}))$ is an execution of $[\![\mathcal{Q}]\!]_\psi$ and $c''_j(g_{i_j}^{-1} \circ \mathbf{e}''_j) = c_{i_j}\,\mathbf{e}_{i_j}$ for all $j \in \{1, \ldots, k-1\}$.

Next, we will construct bijections $g_{m+1}, \ldots, g_n : \mathbb{V} \mapsto \mathbb{V}$ satisfying (i). Let $f_{m+1}, \ldots, f_n$ be bijections: $\mathbb{V} \mapsto \mathbb{V}$ such that for all $j \in \{m+1, \ldots, n\}$, f_j moves only constants in $\psi(T) \setminus (\mathrm{im}(\psi|_{\mathbb{X}}) \cup \mathrm{im}(\mathbf{d}''_{k-1}))$. Now, for each $j \in \{m+1, \ldots, n\}$, let g_j be the bijection $f_j \circ f_{j-1} \circ \ldots \circ f_{m+1} \circ g_m$. By definition, each g_j moves only constants in $\psi(T) \setminus \mathrm{im}(\psi|_{\mathbb{X}})$. By Proposition 29, it means that $(s_{j-1}(g_{j-1} \circ \mathbf{d}_{j-1}), c_j(g_{j-1} \circ \mathbf{e}_j), s_j(g_{j-1} \circ \mathbf{d}_j))$ is a transition of $[\![\mathcal{P}]\!]_\psi$ for all $j \in \{m+1, \ldots, n\}$. Note that we may pick the bijections $f_{m+1}, \ldots, f_n$, in this order, such that for each $j \in \{m+1, \ldots, n\}$, f_j preserves $\mathrm{im}(g_{j-1} \circ \mathbf{d}_{j-1})$. Moreover, since ϕ is a $\{T\}$-subvaluation of ψ and

$$\begin{aligned}|\phi(T)| &\geq \mathrm{idc}([\![\mathcal{P}]\!]_\psi, I_T \setminus \mathrm{im}(\psi|_{\mathbb{X}})) \\ &\quad + \mathrm{idc}([\![\mathcal{Q}]\!]_\phi, I_T \setminus \mathrm{im}(\psi|_{\mathbb{X}})) + |\mathrm{im}(\psi|_{\mathbb{X}}) \cap I_T| \\ &\geq |(\mathrm{im}(g_{j-1} \circ \mathbf{d}_{j-1}\mathbf{e}_j\mathbf{d}_j) \cup \mathrm{im}(\mathbf{d}''_{k-1}) \cup \mathrm{im}(\psi|_{\mathbb{X}})) \cap I_T|\,,\end{aligned}$$

we may also assume that f_j maps $\mathrm{im}(g_{j-1} \circ \mathbf{e}_j\mathbf{d}_j))$ to $\mathrm{Im}(\phi)$ whenever $j \in \{m+1, \ldots, n\}$. By Proposition 29, it implies that $(s_{j-1}(f_j \circ g_{j-1} \circ \mathbf{d}_{j-1}), c_j(f_j \circ g_{j-1} \circ \mathbf{e}_j), s_j(f_j \circ g_{j-1} \circ \mathbf{d}_j)) = (s_{j-1}(g_{j-1} \circ \mathbf{d}_{j-1}), c_j(g_j \circ \mathbf{e}_j), s_j(g_j \circ \mathbf{d}_j))$ is a transition of $[\![\mathcal{P}]\!]_\psi$ for all $j \in \{m+1, \ldots, n\}$. Since $\mathrm{im}(g_i \circ \mathbf{e}_i\mathbf{d}_i) \subseteq \mathrm{Im}(\phi)$, by Lemma 30, $(s_{j-1}(g_{j-1} \circ \mathbf{d}_{j-1}), c_j(g_j \circ \mathbf{e}_j), s_j(g_j \circ \mathbf{d}_j))$ is a transition of $[\![\mathcal{P}]\!]_\phi$ for all $j \in \{m+1, \ldots, n\}$. Hence, $(s_0\,\mathbf{d}_0, c_1(g_1 \circ \mathbf{e}_1), s_1(g_1 \circ \mathbf{d}_1), \ldots, c_n(g_n \circ \mathbf{e}_n), s_n(g_n \circ \mathbf{d}_n))$ is an execution of $[\![\mathcal{P}]\!]_\phi$.

To prove the existence of an execution of $[\![\mathcal{Q}]\!]_\phi$ satisfying (ii), recall that $[\![\mathcal{Q}]\!]_\phi$ does not involve invisible events and $[\![\mathcal{P}]\!]_\phi$ is a trace refinement of $[\![\mathcal{Q}]\!]_\phi$. It means that there is an execution $(s'_0\,\mathbf{d}'_0, c'_1\,\mathbf{e}'_1, s'_1\,\mathbf{d}'_1, \ldots, c'_k\,\mathbf{e}'_k, s'_k\,\mathbf{d}'_k)$ of $[\![\mathcal{Q}]\!]_\phi$ such that $c'_j\,\mathbf{e}'_j = c_{i_j}(g_{i_j} \circ \mathbf{e}_{i_j})$, i.e., $c'_j(g_{i_j}^{-1} \circ \mathbf{e}'_j) = c_{i_j} \circ \mathbf{e}_{i_j}$, for all $j \in \{1, \ldots, k\}$. Since $[\![\mathcal{Q}]\!]_\phi$ is deterministic, the execution $(s'_0\,\mathbf{d}'_0, c'_1\,\mathbf{e}'_1, s'_1\,\mathbf{d}'_1, \ldots, c'_{k-1}\,\mathbf{e}'_{k-1}, s'_{k-1}\,\mathbf{d}'_{k-1})$ is the same as $(s''_0\,\mathbf{d}''_0, c''_1\,\mathbf{e}''_1, s''_1\,\mathbf{d}''_1, \ldots, c''_{k-1}\,\mathbf{e}''_{k-1}, s''_{k-1}\,\mathbf{d}''_{k-1})$, which implies that $(s'_0\,\mathbf{d}'_0, c'_1(g_{i_1}^{-1} \circ \mathbf{e}'_1), s'_1(g_{i_1}^{-1} \circ \mathbf{d}'_1), \ldots, c'_{k-1}(g_m^{-1} \circ \mathbf{e}'_{k-1}), s'_{k-1}(g_m^{-1} \circ \mathbf{d}'_{k-1}))$ is an execution of $[\![\mathcal{Q}]\!]_\psi$. By Lemma 30 and Proposition 29, $(s'_{k-1}(g_n^{-1} \circ \mathbf{d}'_{k-1}), c'_k(g_n^{-1} \circ \mathbf{e}_k), s'_k(g_n^{-1} \circ \mathbf{d}'_k))$ is a transition of $[\![\mathcal{Q}]\!]_\psi$. Since each f_j preserves $\mathrm{im}(\mathbf{d}'_{k-1})$, $g_n^{-1} \circ \mathbf{d}'_{k-1} = g_m^{-1} \circ f_{m+1}^{-1} \circ \ldots \circ f_n^{-1} \circ \mathbf{d}'_{k-1} = g_m^{-1} \circ \mathbf{d}'_{k-1}$, which means that $(s'_{k-1}(g_m^{-1} \circ \mathbf{d}'_{k-1}), c'_k(g_n^{-1} \circ \mathbf{e}_k), s'_k(g_n^{-1} \circ \mathbf{d}'_k))$ is a transition of $[\![\mathcal{Q}]\!]_\psi$. Hence, $(s'_0\,\mathbf{d}'_0, c'_1(g_{i_1}^{-1} \circ \mathbf{e}'_1), s'_1(g_{i_1}^{-1} \circ \mathbf{d}'_1), \ldots, c'_k(g_{i_k}^{-1} \circ \mathbf{e}'_k), s'_k(g_{i_k}^{-1} \circ \mathbf{d}'_k))$ is an execution of $[\![\mathcal{Q}]\!]_\psi$ and the induction step is complete. ∎

Proposition 18: Let us assume that $[\![\mathcal{Q}]\!]_\phi$ is deterministic and $[\![\mathcal{P}]\!]_\phi \preceq_{\mathrm{tr}} [\![\mathcal{Q}]\!]_\phi$. To prove the lemma, we need to show that $\mathrm{alph}([\![\mathcal{P}]\!]_\psi) = \mathrm{alph}([\![\mathcal{Q}]\!]_\psi)$ and $\mathrm{tr}([\![\mathcal{P}]\!]_\psi) \subseteq \mathrm{tr}([\![\mathcal{Q}]\!]_\psi)$.

First, if $c\,\mathbf{e} \in \mathrm{alph}([\![\mathcal{P}]\!]_\psi)$, there are states $s_1\,\mathbf{d}_1, s'_1\,\mathbf{d}'_1$ such that $(s_1\,\mathbf{d}_1, c\,\mathbf{e}, s'_1\,\mathbf{d}'_1)$ is a transition of $[\![\mathcal{P}]\!]_\psi$. Since $|\phi(T)| \geq \mathrm{idc}([\![\mathcal{P}]\!]_\psi, I_T \setminus \mathrm{im}(\psi|_{\mathbb{X}})) + |\mathrm{im}(\psi|_{\mathbb{X}}) \cap I_T| \geq |(\mathrm{im}(\mathbf{d}_1\mathbf{e}\mathbf{d}'_1) \cup \mathrm{im}(\psi|_{\mathbb{X}})) \cap I_T|$, we can pick a bijection $g : \mathbb{V} \mapsto \mathbb{V}$ such that g moves only constants in $\psi(T) \setminus \mathrm{im}(\psi|_{\mathbb{X}})$ and maps those in $\mathrm{im}(\mathbf{d}_1\mathbf{e}\mathbf{d}'_1)$ to $\mathrm{Im}(\phi)$. Then, by Proposition 29 and Lemma 30, $(s_1(g \circ \mathbf{d}_1), c(g \circ \mathbf{e}), s'_1(g \circ \mathbf{d}'_1))$ is a transition of $[\![\mathcal{P}]\!]_\phi$. As $c(g \circ \mathbf{e}) \in \mathrm{alph}([\![\mathcal{P}]\!]_\phi)$ and $[\![\mathcal{P}]\!]_\phi \preceq_{\mathrm{tr}} [\![\mathcal{Q}]\!]_\phi$, $c(g \circ \mathbf{e}) \in \mathrm{alph}([\![\mathcal{Q}]\!]_\phi)$, too. Hence, there are states $s_2\,\mathbf{d}_2, s'_2\,\mathbf{d}'_2$ such that $(s_2\,\mathbf{d}_2, c(g \circ \mathbf{e}), s'_2\,\mathbf{d}'_2)$ is a transition of $[\![\mathcal{Q}]\!]_\phi$. By Lemma 30 and Proposition 29, it means that $(s_2(g^{-1} \circ \mathbf{d}_2), c\,\mathbf{e}, s'_2(g^{-1} \circ \mathbf{d}'_2))$ is a transition of $[\![\mathcal{Q}]\!]_\psi$. Hence, $c\,\mathbf{e} \in \mathrm{alph}([\![\mathcal{Q}]\!]_\psi)$, too. Similarly, one can show that every event in $\mathrm{alph}([\![\mathcal{Q}]\!]_\psi)$ is also an event in $\mathrm{alph}([\![\mathcal{P}]\!]_\psi)$, which implies that the alphabets of $[\![\mathcal{Q}]\!]_\psi$ and $[\![\mathcal{P}]\!]_\psi$ match.

Next, if $t \in \mathrm{tr}([\![\mathcal{P}]\!]_\psi)$, there is an execution $(s_0\,\mathbf{d}_0, c_1\,\mathbf{e}_1, s_1\,\mathbf{d}_1, \ldots, c_n\,\mathbf{e}_n, s_n\,\mathbf{d}_n)$ of $[\![\mathcal{P}]\!]_\psi$ such that t is obtained from $(c_1\,\mathbf{e}_1, \ldots, c_n\,\mathbf{e}_n)$ by erasing the invisible events. We may also assume that either $n = 0$ or $c_n\,\mathbf{e}_n$ is a visible event. By Lemma 17, it means that there is an execution $(s'_0\,\mathbf{d}'_0, c'_1\,\mathbf{e}'_1, s'_1\,\mathbf{d}'_1, \ldots, c'_k\,\mathbf{e}'_k, s'_k\,\mathbf{d}'_k)$ of $[\![\mathcal{Q}]\!]_\psi$

such that $(c'_1\,\mathbf{e}'_1, \ldots, c'_k\,\mathbf{e}'_k)$ equals $(c_1\,\mathbf{e}_1, \ldots, c_n\,\mathbf{e}_n)$ after erasing the invisible events. Therefore, $t \in \mathrm{tr}([\![\mathcal{Q}]\!]_\psi)$, too, which proves that the lemma is correct. ∎

Lemma 19: We argue by induction on the structure of $\mathcal{P}$ by using the lemma as an induction hypothesis.

In the base step, $\mathcal{P}$ is an SPP $(\hat{s}\,\hat{\mathbf{x}}, \Delta)$. Since $\mathrm{im}(\hat{\mathbf{x}}) \subseteq \mathrm{dom}(\phi)$, the set $\mathrm{im}(\phi \circ \hat{\mathbf{x}}) \cap (I_T \setminus \mathrm{im}(\phi|_{\mathbb{X}}))$ is empty. Moreover, it is easy to see that for every $(s\,\mathbf{x}, X, \mathcal{C}, c\,\mathbf{y}, s'\mathbf{x}') \in \Delta$ and $\phi' \in \mathrm{ext}(\phi, \mathrm{im}(\mathbf{x}) \cup X)$ such that $[\![\mathcal{C}]\!]_{\phi'}$ is *true* it holds that

$$\begin{aligned}&|\{x \in \mathrm{im}(\mathbf{x}) \cup X \mid T_x = T\}|\\ &\quad\geq |\{\phi'(x) \in I_T \mid x \in \mathrm{im}(\mathbf{x}) \cup X\}|\\ &\quad\quad\geq |\{\mathrm{im}(\phi' \circ \mathbf{xyx}') \cap (I_T \setminus \mathrm{im}(\phi|_{\mathbb{X}}))\}|\,,\end{aligned}$$

which means that $\mathrm{dbnd}_{T,\mathcal{R}}(\mathcal{P}) \geq \mathrm{idc}([\![\mathcal{P}]\!]_\phi, I_T \setminus \mathrm{im}(\phi|_{\mathbb{X}}))$ for every data type $T \in \mathrm{dom}(\phi)$. Hence, the base step is clear.

In the induction step, there are four cases to consider. First, we assume that $\mathcal{P}$ is $[\mathcal{C}]\mathcal{P}'$. If $[\![\mathcal{C}]\!]_\phi$ is *false*, then $[\![\mathcal{P}']\!]_\phi = P_{id}$. Since $\mathrm{idc}(P_{id}, I_T \setminus \mathrm{im}(\phi|_{\mathbb{X}})) = 0$ for every data type $T \in \mathrm{dom}(\phi)$, it is obvious that the claim holds. Otherwise, $[\![\mathcal{C}]\!]_\phi$ is *true* and $[\![\mathcal{P}]\!]_\phi = [\![\mathcal{P}']\!]_\phi$. Then, by the induction hypothesis,

$$\begin{aligned}\mathrm{dbnd}_{T,\mathcal{R}}(\mathcal{P}) &= \mathrm{dbnd}_{T,\mathcal{R}}(\mathcal{P}')\\ &\geq \mathrm{idc}([\![\mathcal{P}']\!]_\phi, I_T \setminus \mathrm{im}(\phi|_{\mathbb{X}}))\\ &= \mathrm{idc}([\![\mathcal{P}]\!]_\phi, I_T \setminus \mathrm{im}(\phi|_{\mathbb{X}}))\,,\end{aligned}$$

for all data types $T \in \mathrm{dom}(\phi)$, which implies that this case is clear.

Next, we assume that $\mathcal{P}$ is $\mathcal{P}_1 \mathbin{\widehat{\|}} \mathcal{P}_2$. First, note that every state of $[\![\mathcal{P}]\!]_\phi$ consists of a state of $[\![\mathcal{P}_1]\!]_\phi$ and a state of $[\![\mathcal{P}_2]\!]_\phi$. Secondly, note that every transition of $[\![\mathcal{P}]\!]_\phi$ consists of either a transition of $[\![\mathcal{P}_1]\!]_\phi$ and a transition of $[\![\mathcal{P}_2]\!]_\phi$ or a transition of $[\![\mathcal{P}_i]\!]_\phi$ and a state of $[\![\mathcal{P}_j]\!]_\phi$, where i and j are different integers in $\{1, 2\}$. Therefore, it is obvious that

$$\mathrm{idc}([\![\mathcal{P}]\!]_\phi, I_T \setminus \mathrm{im}(\phi|_{\mathbb{X}})) \leq \sum_{i \in \{1,2\}} \mathrm{idc}([\![\mathcal{P}_i]\!]_\phi, I_T \setminus \mathrm{im}(\phi|_{\mathbb{X}}))$$

for every data type $T \in \mathrm{dom}(\phi)$. Therefore, by the induction hypothesis,

$$\begin{aligned}&\mathrm{dbnd}_{T,\mathcal{R}}(\mathcal{P})\\ &\quad= \mathrm{dbnd}_{T,\mathcal{R}}(\mathcal{P}_1) + \mathrm{dbnd}_{T,\mathcal{R}}(\mathcal{P}_2)\\ &\quad\geq \sum_{i \in \{1,2\}} \mathrm{idc}([\![\mathcal{P}_i]\!]_\phi, I_T \setminus \mathrm{im}(\phi|_{\mathbb{X}}))\\ &\quad\quad\geq \mathrm{idc}([\![\mathcal{P}]\!]_\phi, I_T \setminus \mathrm{im}(\phi|_{\mathbb{X}}))\end{aligned}$$

for all data types $T \in \mathrm{dom}(\phi)$, so also this case is clear.

After that, we assume that $\mathcal{P}$ is $\mathcal{P}' \mathbin{\widehat{\setminus}} E$. Since hiding affects only channel names, obviously $\mathrm{idc}([\![\mathcal{P}]\!]_\phi, I_T \setminus \mathrm{im}(\phi|_{\mathbb{X}})) = \mathrm{idc}([\![\mathcal{P}']\!]_\phi, I_T \setminus \mathrm{im}(\phi|_{\mathbb{X}}))$. Then, by the induction hypothesis, it is easy to see that the claim holds in this case, too.

Finally, we consider the case when $\mathcal{P}$ is $\widehat{\|}_x \mathcal{P}'$. By the induction hypothesis, we know that $\mathrm{dbnd}_{T,\mathcal{R}}(\mathcal{P}') \geq \mathrm{idc}([\![\mathcal{P}']\!]_{\phi'}, I_T \setminus \mathrm{im}(\phi'|_{\mathbb{X}}))$ for every $\phi' \in \mathrm{ext}(\phi, \{x\})$ and all data types $T \in \mathrm{dom}(\phi')$. Since

$$\mathrm{cp}_{T_x}(\mathcal{R}) \geq |\phi(T_x)| = |\mathrm{ext}(\phi, \{x\})|\,,$$

it implies that

$$\begin{aligned}\mathrm{dbnd}_{T,\mathcal{R}}(\mathcal{P}) &= \mathrm{cp}_{T_x}(\mathcal{R}) \cdot \mathrm{dbnd}_{T,\mathcal{R}}(\mathcal{P}')\\ &\geq |\mathrm{ext}(\phi, \{x\})| \cdot \mathrm{dbnd}_{T,\mathcal{R}}(\mathcal{P}')\\ &\geq \sum_{\phi' \in \mathrm{ext}(\phi,\{x\})} \mathrm{idc}([\![\mathcal{P}']\!]_{\phi'}, I_T \setminus \mathrm{im}(\phi'|_{\mathbb{X}}))\end{aligned}$$

whenever $T \in \mathrm{dom}(\phi)$ is a data type. Now, we argue like in the second case of the induction step that every constant in a state of $[\![\mathcal{P}]\!]_\phi$ occurs in a state of some $[\![\mathcal{P}']\!]_{\phi'}$, where $\phi' \in \mathrm{ext}(\phi, \{x\})$, and every constant in a transition of $[\![\mathcal{P}]\!]_\phi$ occurs in a state or a transition of some $[\![\mathcal{P}']\!]_{\phi'}$, where again $\phi' \in \mathrm{ext}(\phi, \{x\})$. Moreover, since x is a process variable, it is obvious that $I_T \setminus \mathrm{im}(\phi'|_{\mathbb{X}}) = I_T \setminus \mathrm{im}(\phi|_{\mathbb{X}})$ for all data types $T \in \mathrm{dom}(\phi)$. Therefore,

$$\begin{aligned}&\sum_{\phi' \in \mathrm{ext}(\phi,\{x\})} \mathrm{idc}([\![\mathcal{P}']\!]_{\phi'}, I_T \setminus \mathrm{im}(\phi'|_{\mathbb{X}}))\\ &\quad\quad\geq \mathrm{idc}([\![\mathcal{P}]\!]_\phi, I_T \setminus \mathrm{im}(\phi|_{\mathbb{X}}))\,,\end{aligned}$$

for every data type $T \in \mathrm{dom}(\phi)$, which completes the induction step. Hence, by the induction principle, the lemma is correct. ∎

Theorem 20: Since $\Phi \subseteq \Psi$, it obvious that when $[\![\mathcal{P}]\!]_\psi \preceq_{\mathrm{tr}} [\![\mathcal{Q}]\!]_\psi$ for all valuations $\psi \in \Psi$, then $[\![\mathcal{P}]\!]_\phi \preceq_{\mathrm{tr}} [\![\mathcal{Q}]\!]_\phi$ for all valuations $\phi \in \Phi$, too.

Next, let us assume that $[\![\mathcal{P}]\!]_\phi \preceq_{\mathrm{tr}} [\![\mathcal{Q}]\!]_\phi$ for all valuations $\phi \in \Phi$ and let $\psi \in \Psi$. To prove the theorem, it is sufficient to show that then $[\![\mathcal{P}]\!]_\psi \preceq_{\mathrm{tr}} [\![\mathcal{Q}]\!]_\psi$, too. First, we pick data types $T_1, \ldots, T_n \in \overline{\mathrm{par}}(\mathcal{P} \mathbin{\widehat{\|}} \mathcal{Q})$ and valuations $\psi_0, \ldots, \psi_n \in \Psi$ such that $\psi_0 \in \Phi$, $\psi_n = \psi$ and for all $i \in \{1, \ldots, n\}$, ψ_{i-1} is a $\{T_i\}$-subvaluation of ψ_i and $|\psi_{i-1}(T_i)| = \mathrm{cd}_{T_i}(\mathcal{P} \mathbin{\widehat{\|}} \mathcal{Q})$. After that, we proceed by induction to show that $[\![\mathcal{P}]\!]_{\psi_i} \preceq_{\mathrm{tr}} [\![\mathcal{Q}]\!]_{\psi_i}$ for all $i \in \{0, \ldots, n\}$.

The base step is clear since $\phi_0 \in \Phi$ and the claim holds by the assumption. In the induction step, we apply the definition and Lemma 19 to see that

$$\begin{aligned}&|\psi_{i-1}(T_i)| = \mathrm{cd}_{T_i}(\mathcal{P} \mathbin{\widehat{\|}} \mathcal{Q})\\ &\quad= \max\big(1, |\mathrm{free}_{T_i}(\mathcal{P}) \cup \mathrm{free}_{T_i}(\mathcal{Q})|\\ &\quad\quad+ \mathrm{dbnd}_{T_i,\mathcal{P}\widehat{\|}\mathcal{Q}}(\mathcal{P}) + \mathrm{dbnd}_{T_i,\mathcal{P}\widehat{\|}\mathcal{Q}}(\mathcal{Q})\big)\\ &\quad\geq |\{x \in \mathrm{dom}(\psi_i|_{\mathbb{X}}) \mid T_x = T_i\}|\\ &\quad\quad+ \mathrm{idc}([\![\mathcal{P}]\!]_{\psi_i}, I_{T_i} \setminus \mathrm{im}(\psi_i|_{\mathbb{X}}))\\ &\quad\quad+ \mathrm{idc}([\![\mathcal{Q}]\!]_{\psi_i}, I_{T_i} \setminus \mathrm{im}(\psi_i|_{\mathbb{X}}))\end{aligned}$$

$$\begin{aligned}\geq & |\mathrm{im}(\psi_i|_{\mathbb{X}}) \cap I_{T_i}| \\ & + \mathrm{idc}([\![\mathcal{P}]\!]_{\psi_i}, I_{T_i} \setminus \mathrm{im}(\psi_i|_{\mathbb{X}})) \\ & + \mathrm{idc}([\![\mathcal{Q}]\!]_{\psi_i}, I_{T_i} \setminus \mathrm{im}(\psi_i|_{\mathbb{X}}))\ .\end{aligned}$$

Since $[\![\mathcal{Q}]\!]_{\psi_{i-1}}$ is deterministic and, by the induction hypothesis, $[\![\mathcal{P}]\!]_{\psi_{i-1}} \preceq_{\mathrm{tr}} [\![\mathcal{Q}]\!]_{\psi_{i-1}}$, it follows from Proposition 18 that $[\![\mathcal{P}]\!]_{\psi_i} \preceq_{\mathrm{tr}} [\![\mathcal{Q}]\!]_{\psi_i}$, too. Hence, by the induction principle, $[\![\mathcal{P}]\!]_{\psi_i} \preceq_{\mathrm{tr}} [\![\mathcal{Q}]\!]_{\psi_i}$ for all $i \in \{0, \ldots, n\}$, which means that especially $[\![\mathcal{P}]\!]_{\psi} \preceq_{\mathrm{tr}} [\![\mathcal{Q}]\!]_{\psi}$ holds. ■

Lemma 31: To prove that Φ is finite, assume that it is not. Since all the valuations in Φ share the same finite domain and $|\phi(T)| \leq k$ for all $\phi \in \Phi$ and for all types $T \in \overline{\mathrm{par}}(\mathcal{P} \mathbin{\widehat{\|}} \mathcal{Q})$, there is an infinite subset Φ' of Φ such that all valuations in Φ' map each type in the domain to a set of the same size, i.e., $|\phi_1(T)| = |\phi_2(T)|$ for all $\phi_1, \phi_2 \in \Phi'$ and for all types $T \in \overline{\mathrm{par}}(\mathcal{P} \mathbin{\widehat{\|}} \mathcal{Q})$. It means that for every valuation $\phi' \in \Phi'$, we can pick a bijection $g_{\phi'} : \mathbb{V} \mapsto \mathbb{V}$ such that $(g_{\phi_1} \circ \phi_1)(T) = (g_{\phi_2} \circ \phi_2)(T)$ for all $\phi_1, \phi_2 \in \Phi'$ and for all types $T \in \overline{\mathrm{par}}(\mathcal{P} \mathbin{\widehat{\|}} \mathcal{Q})$. Since we assumed the valuations in Φ to be non-isomorphic, a set $\Phi'' := \{g_{\phi'} \circ \phi' \mid \phi' \in \Phi'\}$ must be infinite. However, this is impossible, since each valuation $\phi'' \in \Phi''$ covers the same finite set of variables and the values of the variables are always picked from the same finite set $\mathrm{Im}(\phi'')$. Therefore, we conclude that Φ must be finite. ■

Proposition 32: By definition, there is a bijection $g : \mathbb{V} \mapsto \mathbb{V}$ such that $\psi = g \circ \phi$. Necessarily, g preserves the constants in I_T for every type T. Now, if $[\![\mathcal{P}]\!]_{\phi} \preceq_{\mathrm{tr}} [\![\mathcal{Q}]\!]_{\phi}$, then by Item 10 of Proposition 23, $g([\![\mathcal{P}]\!]_{\phi}) \preceq_{\mathrm{tr}} g([\![\mathcal{Q}]\!]_{\phi})$, too. By Lemma 28, it means that $[\![\mathcal{P}]\!]_{g\circ\phi} \preceq_{\mathrm{tr}} [\![\mathcal{Q}]\!]_{g\circ\phi}$ or, in other words, $[\![\mathcal{P}]\!]_{\psi} \preceq_{\mathrm{tr}} [\![\mathcal{Q}]\!]_{\psi}$. ■

Corollary 21: By Lemma 31, the finiteness of Φ is clear. Moreover, if $\mathcal{P} \mathbin{\widehat{\preceq}}_{\mathrm{tr}} \mathcal{Q}$, then by definition, $[\![\mathcal{P}]\!]_{\phi} \preceq_{\mathrm{tr}} [\![\mathcal{Q}]\!]_{\phi}$ for all $\phi \in \Phi$, too. To prove the other direction, let Ψ be the set of all valuations with the domain $\overline{\mathrm{par}}(\mathcal{P} \mathbin{\widehat{\|}} \mathcal{Q})$, Ψ' the set of all valuations $\psi \in \Psi$ such that $|\psi(U)| \leq \mathrm{cp}_U(\mathcal{P} \mathbin{\widehat{\|}} \mathcal{Q})$ for every process type $U \in \mathrm{dom}(\psi)$ and Ψ'' the set of all valuations $\psi' \in \Psi'$ such that $|\psi'(T)| \leq \mathrm{cd}_T(\mathcal{P} \mathbin{\widehat{\|}} \mathcal{Q})$ for every data type $T \in \mathrm{dom}(\psi')$. Now, if $[\![\mathcal{P}]\!]_{\phi} \preceq_{\mathrm{tr}} [\![\mathcal{Q}]\!]_{\phi}$ for all $\phi \in \Phi$, then by Proposition 32, $[\![\mathcal{P}]\!]_{\psi''} \preceq_{\mathrm{tr}} [\![\mathcal{Q}]\!]_{\phi''}$ for all $\psi'' \in \Psi''$. By Theorem 20, it implies that then $[\![\mathcal{P}]\!]_{\psi'} \preceq_{\mathrm{tr}} [\![\mathcal{Q}]\!]_{\psi'}$ for all $\psi' \in \Psi'$, and by Theorem 16, we see that $[\![\mathcal{P}]\!]_{\psi} \preceq_{\mathrm{tr}} [\![\mathcal{Q}]\!]_{\psi}$ for all $\psi \in \Psi$, too. Finally, by Proposition 24, it means that then $[\![\mathcal{P}]\!]_{\phi} \preceq_{\mathrm{tr}} [\![\mathcal{Q}]\!]_{\phi}$ for all compatible valuations ϕ, in other words, $\mathcal{P} \mathbin{\widehat{\preceq}}_{\mathrm{tr}} \mathcal{Q}$. ■

Theorem 22: Similar to the proof of Corollary 21 but simpler. ■

C. Figures

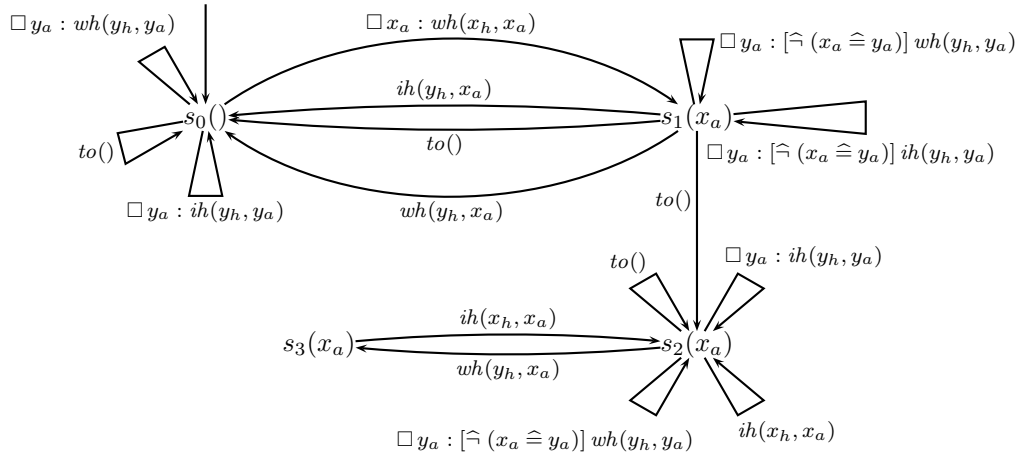

Figure 3. SPP $Host(x_h, y_h, T_A)$ representing the behaviour of HCP from the viewpoint of two hosts, where $to()$ denotes timeout and $wh(z_h, z_a)$ means that the host z_a wants to know whether someone has the address z_a.

D. Verification Models

1) HCP:

```
type H
type A

var h : H
var h2: H
var a : A
var a2: A

chan timeout
chan whohas : H,A
chan ihave : H,A

plts Host =
    lts
        I =    timeout() -> I
            [] []a:whohas(h2,a) -> I
            [] []a:ihave(h2,a) -> I
            [] []a:whohas(h,a) -> W(a)
        W(a) =    timeout() -> S(a)
            [] timeout() -> I
            [] ihave(h2,a) -> I
            [] whohas(h2,a) -> I
            [] []a2:[!a2=a] ihave(h2,a2) -> W(a)
            [] []a2:[!a2=a] whohas(h2,a2) -> W(a)
        S(a) =      timeout() -> S(a)
            [] ihave(h,a) -> S(a)
            [] whohas(h2,a) -> R(a)
            [] []a2:[!a2=a] whohas(h2,a2) -> S(a)
            [] []a2:ihave(h2,a2) -> S(a)
        R(a) =     ihave(h,a) -> S(a)
    from I
```

```
plts DifAdr =
    lts
        I =          []a2:ihave(h2,a2) -> I
            [] []a: ihave(h,a) -> S1(a)
        S1(a) =      []a2:[!a2=a] ihave(h2,a2) -> S1(a)
            [] ihave(h,a) -> S1(a)
    from I

pset WTEv = (_)h,a:{timeout(), whohas(h,a)}

plts Sys = ||h,h2:[!h=h2] Host

plts Spec = ||h,h2:[!h=h2] DifAdr

verify Sys \ WTEv against Spec wrt traces when true
```

2) SRS:

```
type U
type R
type D

avar k : U
avar k1 : U
avar k2 : U
avar r : R
avar d : D

chan rdlock : U, R
chan wrlock : U, R
chan unlock : U, R
chan rdbeg : U, R
chan rdend : U, R, D
chan wrbeg : U, R, D
chan wrend : U, R

ltsc User =
    lts
        I =    rdlock(k,r) -> R
            [] wrlock(k,r) -> W
            [] tau() -> I
        R =    rdbeg(k,r) -> R2
            [] wrlock(k,r) -> W
            [] unlock(k,r) -> I
        R2 = [] d : rdend(k,r,d) -> R
        W =    rdbeg(k,r) -> W2
            [] [] d : wrbeg(k,r,d) -> W3
            [] unlock(k,r) -> I
        W2 = [] d : rdend(k,r,d) -> W
        W3 =   wrend(k,r) -> W
    from I

ltsc Lock =
    lts
```

```
        NO =   rdlock(k1,r) -> R1
            [] wrlock(k1,r) -> W1
            [] rdlock(k2,r) -> R2
            [] wrlock(k2,r) -> W2
        R1 =   wrlock(k1,r) -> W1
            [] unlock(k1,r) -> NO
            [] rdlock(k2,r) -> R12
        R2 =   wrlock(k2,r) -> W2
            [] unlock(k2,r) -> NO
            [] rdlock(k1,r) -> R12
        W1 =   unlock(k1,r) -> NO
        W2 =   unlock(k2,r) -> NO
        R12 =  unlock(k1,r) -> R2
            [] unlock(k2,r) -> R1
    from NO

ltsc Mutex1 =
    lts
        NO =   rdbeg(k,r) -> R1
            [] [] d : wrbeg(k,r,d) -> N1
        N1 =   wrend(k,r) -> NO
        R1 =   [] d : rdend(k,r,d) -> NO
    from NO

ltsc Mutex2 =
    lts
        NO =   rdbeg(k1,r) -> R1
            [] rdbeg(k2,r) -> R2
            [] [] d : wrbeg(k1,r,d) -> N1
            [] [] d : wrbeg(k2,r,d) -> N2
        N1 =     wrend(k1,r) -> NO
        N2 =     wrend(k2,r) -> NO
        R1 =   [] d : rdend(k1,r,d) -> NO
            [] rdbeg(k2,r) -> RR
        R2 =   [] d : rdend(k2,r,d) -> NO
            [] rdbeg(k1,r) -> RR
        RR =   [] d : rdend(k1,r,d) -> R2
            [] [] d : rdend(k2,r,d) -> R1
    from NO

ltsc SRS = (|| k, r : User) || (|| r, k1, k2 : [!k1=k2] Lock)

ltsc Mutex = (|| k1, k2, r : [!k1=k2] Mutex2) || (|| k,r : Mutex1)

ssc LcEv = (_) k, r : {rdlock(k,r), wrlock(k,r), unlock(k,r)}

verify SRS \ LcEv against Mutex wrt traces when true
```

3) Cache:

```
type P
type R
type D

var p : P
```

```
var p2 : P
var r : R
var r2 : R
var d : D
var d2 : D

chan rdbeg : P,R
chan rdend : P,R,D
chan wrbeg : P,R,D
chan wrend : P,R
chan rdmem : P,R,D
chan flush : R,D

plts CUnit =
    lts
        I =           rdbeg(p,r) -> IR
            [] [] d : wrbeg(p,r,d) -> IW(d)
            []        rdbeg(p2,r) -> I
            [] [] d : wrbeg(p2,r,d) -> I
            [] [] d : flush(r,d) -> I
        IR =   [] d : rdmem(p,r,d) -> IR2(d)
            []        rdbeg(p2,r) -> IR
        IR2(d) =      rdend(p,r,d) -> S(d)
            []        rdbeg(p2,r) -> IR2(d)
        IW(d) =       wrend(p,r) -> M(d)

        S(d) =     rdbeg(p,r) -> SR(d)
            []     rdbeg(p2,r) -> S(d)
            [] [] d : wrbeg(p,r,d) -> SW(d)
            []     flush(r,d) -> I
            []     tau() -> I
        SR(d) =       rdend(p,r,d) -> S(d)
            []        rdbeg(p2,r) -> SR(d)
        SW(d) =       wrend(p,r) -> M(d)

        M(d) =        rdbeg(p,r) -> MR(d)
            [] [] d : wrbeg(p,r,d) -> MW(d)
            []     flush(r,d) -> S(d)
            []     flush(r,d) -> I
        MR(d) =       rdend(p,r,d) -> M(d)
        MW(d) =       wrend(p,r) -> M(d)
    from I

plts Cache = || p,p2,r : [!p=p2] CUnit

plts MUnit =
    lts
        I = tau() -> M(d)
        M(d) =            rdmem(p,r,d) -> M(d)
               [] [] d2 : flush(r,d2) -> M(d2)
    from I

plts Mem = || p,r : MUnit
```

```
plts CacheMem = Cache || Mem

plts Mut2 =
    lts
        I =    rdbeg(p,r) -> R1
            [] rdbeg(p2,r) -> R2
            [] [] d : wrbeg(p,r,d) -> W1
            [] [] d : wrbeg(p2,r,d) -> W2
        R1 =   [] d : rdend(p,r,d) -> I
            []         rdbeg(p2,r) -> R12
        R2 =   [] d : rdend(p2,r,d) -> I
            []         rdbeg(p,r) -> R12
        R12 =  [] d : rdend(p,r,d) -> R2
            [] [] d : rdend(p2,r,d) -> R1
        W1 =        wrend(p,r) -> I
        W2 =        wrend(p2,r) -> I
    from I

plts Mutex = || p,p2,r : [!p=p2] Mut2

pset MemAcc = (_) p,r,d : {rdmem(p,r,d), flush(r,d)}

verify CacheMem \ MemAcc against Mutex wrt traces when true
```