CSP and Kripke structures

Ana Cavalcanti[1], Wen-ling Huang[2], Jan Peleska[2], and Jim Woodcock[1]

[1] University of York
[2] University of Bremen

Abstract. A runtime verification technique has been developed for CSP via translation of CSP models to Kripke structures. With this technique, we can check that a system under test satisfies properties of traces and refusals of its CSP model. This complements analysis facilities available for CSP and for all languages with a CSP-based semantics: Safety-Critical Java, Simulink, SysML, and so on. Soundness of the verification depends on the soundness of the translation and on the traceability of the Kripke structure analysis back to the CSP models and to the property specifications. Here, we present a formalisation of soundness by unifying the semantics of the languages involved: normalised graphs used in CSP model checking, action systems, and Kripke structures. Our contributions are the unified semantic framework and the formal argument itself.

Keywords: semantic models, UTP, formal testing, runtime verification

1 Introduction

CSP [19] is a well established process algebra with consistent denotational, operational and axiomatic semantics that have been thoroughly studied. A commercial model checker, FDR3 [9] and its predecessors, has been in widespread use for years and has encouraged industrial take up. For finite processes, FDR3 provides a semantics in terms of normalised graphs: deterministic finite automata with edges labelled by events and nodes by sets of maximal refusals.

Recently, this semantics has been used to develop a runtime verification technique for CSP [17]. It checks the behaviour of programs or simulations during their execution against given specifications; this is typically applied in situations where model checking would be infeasible due to the size of the state space.

In the approach of [17], a specification of traces and refusals of a CSP process is translated to a safety LTL formula. Runtime verification of the resulting property is then carried out using a technique that assumes that the system under test (SUT) behaves like an unknown Kripke structure. (Although the technique does not require the construction of the Kripke structure, its soundness is established in terms of an unknown Kripke structure that models the SUT.) Soundness of the CSP technique is argued via translation of the FDR3 normalised graphs to nondeterministic programs, and then to Kripke structures.

Based on the Kripke structures, we can apply an existing runtime verification technique that defines practical health monitors (error-detection mechanisms) [13]. They do not provide false positives or negatives and can be activated at any time during the execution of an SUT. Using the technique in [17], health monitors can be created based on specifications of CSP processes and, therefore, based on any language for which a CSP-based semantics exists. Some very practical examples are Safety-Critical Java [8], Simulink [5], and SysML [15]. For Safety-Critical Java, this technique can complement assertion-based analysis techniques that use JML [3] and SafeJML [10], which support reasoning about data models and execution time, with facilities to reason about reactivity.

Soundness is rigorously argued in [17] based on the following premises:

1. the semantics of finite CSP processes as a normalised graph, as originally described in [18, Chapter 21] and then implemented in FDR3, is consistent with the CSP semantics;
2. a mapping of the normalised graphs into nondeterministic programs defined in [17] preserves the semantics of CSP;
3. a semantics in terms of Kripke structures for these nondeterministic programs, defined in [17] in terms of their operational semantics, preserves the semantics of the programs; and
4. a mapping of a safety LTL formula of a particular form to a trace and refusal specification defined in [17] captures the semantics of the safety formula in the failures model.

With these results, we can then conclude that the notion of satisfaction in the failures model corresponds to the notion of satisfaction in Kripke structures.

In this paper, we still take (1) as a premise: it is widely accepted and validated both in the standard semantic theories of CSP [19] and in the extensive use of FDR3 (and its predecessors). We, however, go further and formalise the notions of semantics preservation in (2) and (3). We carry out this work using Hoare and He's Unifying Theories of Programming [12], a relational semantic framework that allows us to capture and relate theories for a variety of programming paradigms. A UTP theory for CSP is already available, as are many others (for object-orientation [21], time [20], and so on). Finally, as pointed out in [17], (4) is trivial because the mapping from the safety LTL formula subset under consideration to trace and refusal specifications is very simple.

In formalising (2) and (3), we define UTP theories for normalised graphs and Kripke structures. The nondeterministic programs are action systems and are encoded in the UTP theory for reactive processes. Galois connections between these theories establish semantic preservation. Unification is achieved via an extra UTP theory that captures a kind of stable-failures model, where traces are associated with maximal refusals. Galois connections with this extra theory identify the traces and maximal refusals of a normalised graph, an action system, and a Kripke structure. Figure 1 gives an overview of our results.

In the unified context of the theory of traces and maximal refusals, we define satisfaction for CSP normalised graphs and for Kripke structures. The properties that we consider are the conditions, that is, predicates on a single state, of that theory of traces and maximal refusals. The Galois connections are used to establish the relationship between satisfaction in CSP and in Kripke structures.

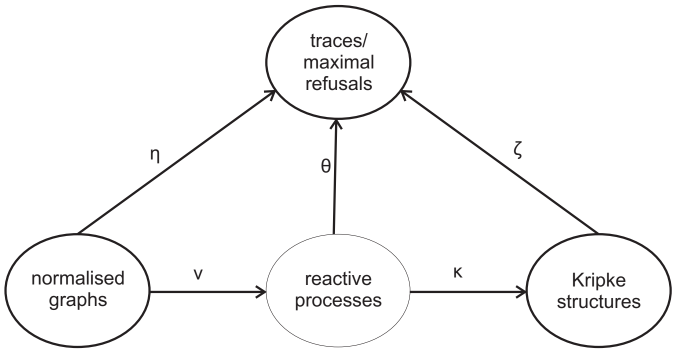

Fig. 1. New UTP theories and their relation to reactive processes

Besides contributing to the UTP agenda of unification of programming theories, we open the possibility of using the runtime verification technique of Kripke structures for other languages with a UTP semantics, such as, *Circus* [16], rCOS [14], Handel-C [4], and SystemC [22].

The approach is restricted to the, still significant, class of divergence-free programs. Divergence freedom is a standard assumption in testing techniques, where observation of divergence is perceived as deadlock.

Next, we give an overview of the UTP and the existing theory of reactive processes. Our theories are presented afterwards: normalised graphs in Section 3, Kripke structures in Section 4, and traces and maximal refusals in Section 5. Section 3 also gives the Galois connection between graphs and reactive processes, and Section 4 between reactive processes and Kripke structures. Finally, Section 5 gives the Galois connections between graphs, reactive processes and Kripke structures and traces and maximal refusals. In Section 5, we also define satisfaction and present our main result: soundness of the CSP runtime verification technique. We conclude and present related and future work in Section 6.

2 A UTP theory of reactive processes

In the UTP, relations are defined by predicates over an alphabet (set) of observational variables that record information about the behaviour of a program. In the simplest theory of general relations, these are the programming variables v, and their dashed counterparts v', with v used to refer to an initial observation of the value of v, and v' to a later observation.

Theories are characterised by an alphabet and by healthiness conditions defined by monotonic idempotent functions from predicates to predicates. The predicates of a theory with alphabet a are the predicates on a that are fixed points of the healthiness conditions. As an example, we consider the existing theory of reactive processes used in our work to model action systems.

A reactive process interacts with its environment: its behaviour cannot be characterised by the relation between its initial and final states only; we need

$$\begin{aligned}\mathsf{R1}(P) &\mathrel{\widehat=} P \wedge tr \le tr'\\ \mathsf{R2}(P) &\mathrel{\widehat=} P[\langle\rangle/tr, (tr' - tr)/tr']\\ \mathsf{R3}(P) &\mathrel{\widehat=} (\mathit{II} \lhd wait \rhd P)\end{aligned}$$

Table 1. Healthiness conditions of the theory of reactive processes

to record information about the intermediate interactions. To that end, the alphabet of the theory of reactive processes includes four extra observational variables: *ok*, *wait*, *tr*, and *ref* and their dashed counterparts.

The variable *ok* is a boolean that records whether the previous process has diverged: *ok* is true if it has not diverged. Similarly, ok' records whether the process itself is diverging. The variable *wait* is also boolean; *wait* records whether the previous process terminated, and $wait'$ whether the process has terminated or not. The purpose of *tr* is to record the trace of events observed so far. Finally, *ref* records a set of events refused, previously (*ref*) or currently (ref').

The monotonic idempotents used to define the healthiness conditions for reactive processes are in Table 1. The first healthiness condition $\mathsf{R1}$ is characterised by the function $\mathsf{R1}(P) \mathrel{\widehat=} P \wedge tr \le tr'$. Its fixed points are all predicates that ensure that the trace of events tr' extends the previously observed trace tr. $\mathsf{R2}$ requires that P is unaffected by the events recorded in tr, since they are events of the previous process. Specifically, $\mathsf{R2}$ requires that P is not changed if we substitute the empty sequence $\langle\rangle$ for tr and the new events in tr', that is, the subsequence $tr' - tr$, for tr'. Finally, the definition of $\mathsf{R3}$ uses a conditional. It requires that, if the previous process has not terminated (*wait*), then a healthy process does not affect the state: it behaves like the identity relation II.

The theory of reactive processes is characterised by the healthiness condition $\mathsf{R} \mathrel{\widehat=} \mathsf{R1} \circ \mathsf{R2} \circ \mathsf{R3}$. The reactive processes that can be described using CSP can be expressed by applying R to a design: a pre and postcondition pair over *ok*, *wait*, *tr* and *ref*, and their dashed counterparts. In such a process $\mathsf{R}(pre \vdash post)$, the precondition *pre* defines the states in which the process does not diverge, and *post* the behaviour when the previous process has not diverged and *pre* holds.

Typically, a theory defines a number of programming operators of interest. Common operators like assignment, sequence, and conditional, are defined for general relations. Sequence is relational composition.

$$P;\ Q \mathrel{\widehat=} \exists\, w_0 \bullet P[w_0/w'] \wedge Q[w_0/w], \textbf{ where } out\alpha(Q) = in\alpha(Q)' = w'$$

The relation $P;\ Q$ is defined by a quantification that relates the intermediate values of the variables. It is required that the set of dashed variables $out\alpha(P)$ of P, named w', matches the undashed variables $in\alpha(Q)$ of Q. The sets w, w', and w_0 are used as lists that enumerate the variables of w and the corresponding decorated variables in the same order.

A central concern of the UTP is refinement. A program P is refined by a program Q, which is written $P \sqsubseteq Q$, if, and only if, $P \Leftarrow Q$, for all possible values of the variables of the alphabet. We write $[P \Leftarrow Q]$ to represent the

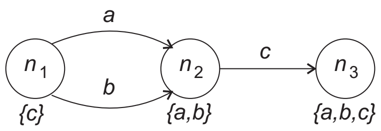

Fig. 2. Normalised graph for $a \rightarrow c \rightarrow STOP \ \Box \ b \rightarrow c \rightarrow STOP$

universal quantification over all variables in the alphabet. The set of alphabetised predicates in the theory of relations form a complete lattice with this ordering.

As well as characterising a set of healthy predicates via their fixed points, healthiness conditions can be viewed as functions from arbitrary relations to predicates of the theory that they define. Since they are monotonic idempotents, their images, that is, the theory that they characterise, are also complete lattices under refinement. In these theories, recursion is modelled by weakest fixed points $\mu X \bullet F(X)$, where F is a monotonic function from predicates to predicates.

In presenting our theories in the next sections, we define their alphabet and healthiness conditions, and prove that the healthiness conditions are monotonic and idempotent. Finally, we establish Galois connections between them.

3 A UTP theory for normalised graphs

A normalised graph $(N, n_0, t : N \times \Sigma \nrightarrow N, r : N \rightarrow \mathbb{F}(\mathbb{F}\,\Sigma))$ is a quadruple, where N is a set of nodes, n_0 is the initial node, t defines the transitions between nodes from N labelled with events from a set Σ, and r defines labels for states as sets of (maximal) refusal sets, that is, finite sets of finite sets of events in Σ.

Alphabet We take N and Σ as global constants, and define the alphabet to contain the variables $n, n' : N$ to represent the source and target nodes, $e : \Sigma^{\varepsilon}$ to represent labels of the transitions, and $r' : \mathbb{F}(\mathbb{F}\,\Sigma^{\varepsilon})$ to represent labels of the target nodes. The predicates define a graph by identifying the source and target nodes n and n' of the transitions, their associated events e, and the labelling r' of the target nodes. The initial node is always ι, a constant of type N. In ι, only the special event ε is available. The set $\Sigma^{\varepsilon} = \Sigma \cup \{\varepsilon\}$.

Example 1. We consider the graph for $a \rightarrow c \rightarrow STOP \ \Box \ b \rightarrow c \rightarrow STOP$ shown in Figure 2 . It is defined in our UTP theory by the following relation.

$$\begin{aligned} EG \mathrel{\widehat=}\ & n = \iota \land e = \varepsilon \land n' = n_1 \land r' = \{\{c, \varepsilon\}\} \lor \\ & n = n_1 \land e \in \{a, b\} \land n' = n_2 \land r' = \{\{a, b, \varepsilon\}\} \lor \\ & n = n_2 \land e = c \land n' = n_3 \land r' = \{\{a, b, c, \varepsilon\}\} \end{aligned}$$

We observe that n_1, n_2 and n_3 are arbitrary node identifiers: values in N. $\Box$

By including just r' in the alphabet, instead of r and r', we avoid the need to specify labelling for a node repeatedly whenever it is used as a source or a target of a transition (and to include a healthiness condition to ensure that the

$$
\begin{array}{ll}
\mathsf{HG1}(G) \mathrel{\widehat=} G \land r' \neq \emptyset & \\
\mathsf{HG2}(G) \mathrel{\widehat=} G \land (n = \iota \Rightarrow e = \varepsilon) \land \forall X : r' \bullet \varepsilon \in X \land n' \neq \iota & \\
\mathsf{HG3}(G) \mathrel{\widehat=} G \land DetEdges(G) \quad DetEdges(G) \mathrel{\widehat=} \forall n, e \bullet \#\{n', r' \mid G \bullet n'\} \leq 1 & \\
\mathsf{HG4}(G) \mathrel{\widehat=} G \land DetRefs(G) \quad DetRefs(G) \mathrel{\widehat=} \forall n' \bullet \#\{n, e, r' \mid G \bullet r'\} \leq 1 & \\
\mathsf{HG5}(G) \mathrel{\widehat=} G \land AccEvents(G) & \\
\qquad AccEvents(G) \mathrel{\widehat=} \forall e_1 \bullet (\forall n'_1, r'_1 \bullet \neg\, G(n', e_1, n'_1, r'_1)) \Rightarrow \forall X : r' \bullet e_1 \in X & \\
\mathsf{HG6}(G) \mathrel{\widehat=} G \land \exists n, e, n', r' \bullet G &
\end{array}
$$

Table 2. Healthiness conditions of the normalised-graph theory

duplicated information is consistent). Since the initial node is always ι, for which labelling is irrelevant, it is enough to define the labels of the target nodes to get information for all (reachable) nodes. Normalised graphs are connected.

Healthiness conditions Table 2 defines the healthiness conditions of our theory. HG1 requires all nodes to have a non-empty label: every label contains at least one set X, and, as specified by HG2, each X contains ε. HG2 is concerned with ι and ε; from ι, the only possible event is ε, which is then always refused, and, besides, no transition leads back to ι. HG3 requires that, for any node n and event e, there is at most one transition: the graph is deterministic. Similarly, HG4 establishes that all transitions that target a node n' define the same label: labelling is unique. HG5 requires that, if there is no transition from a node n' for an event e_1, then e_1 is in all refusals X of the label r' of n'. We write $G(w, x, y, z)$ to denote the predicate $G[w, x, y, z/n, e, n', r']$. Finally, HG6 rules out the empty graph *false*.

All of HG1 to HG6 are conjunctive (that is, of the form $\mathsf{HC}(P) \mathrel{\widehat=} P \land F(P)$, for a function $F(P)$ that is monotonic or does not depend on P). So, they are all monotonic, idempotent, and commute [11]. Commutativity establishes independence of the healthiness conditions. We can then define the healthiness condition HG of our theory as the composition of HG1 to HG6. Commutativity implies that HG is an idempotent, just like each of HG1 to HG6.

Connection to reactive processes In [17], graphs are transformed to nondeterministic programs of a particular form. They are action systems [2]: initialised nondeterministic loops, with part of the state at the beginning of each iteration visible. These are, therefore, reactive processes, that communicate to the environment the value of the relevant state components at the start of a loop.

For a graph G, the corresponding action system $AS(G)$ in [17] is as follows.

Definition 1.

$$
\begin{array}{l}
AS(G) \mathrel{\widehat=} \mathbf{var}\, n, tr, ref \bullet n, tr, ref := \iota, \langle\rangle, \Sigma;\ \mu\, Y \bullet vis!(tr, ref) \rightarrow \\
\qquad Skip \lhd ref = \Sigma^{\varepsilon} \rhd \sqcap\, e : \Sigma^{\varepsilon} \setminus ref \bullet \\
\qquad\qquad tr := tr \frown \langle e \rangle; \\
\qquad\qquad n, ref : [true, \exists\, r' \bullet G \land ref' \in r'];\ \ Y
\end{array}
$$

The program uses a local variable n as a pointer to the current node as it iterates over G. The initial value of n is ι. As the loop progresses, the program accumulates the traces of events tr and records a refusal ref in r. Their values are initialised with the empty sequence $\langle\rangle$ and the whole set of events Σ (but not ε). The values of tr and ref are communicated in each step of the iteration via a channel vis. It is the values that can be communicated that capture the traces and maximal refusals semantics of G.

The loop is defined by a tail recursion ($\mu\, Y \bullet \ldots;\ Y$). Its termination condition is $ref = \Sigma^{\varepsilon}$, that is, it terminates when there is a deadlock. Otherwise, it chooses nondeterministically ($\sqcap$) an event e that can be offered, that is, an event from $\Sigma^{\varepsilon} \setminus ref$, updates tr to record that event, and then updates n and ref as defined by G using a design $n, ref : [true, \exists\, r' \bullet G \wedge ref' \in r']$. The postcondition $\exists\, r' \bullet G \wedge ref' \in r'$ defines the new values of n and ref; the value of ref is also chosen nondeterministically from the events in r' as defined by G.

Example 2. For the process in Example 1, the corresponding reactive process obtained from the graph in Figure 2, is equivalent to that shown below, where we unfold the recursion and eliminate nondeterministic choices over one element relying on the property $(\sqcap\, e_1 : \{e_2\} \bullet P(e_1)) = P(e_2)$.

$$\begin{array}{l}
\mathbf{var}\ n, tr, ref \bullet n, tr, ref := \iota, \langle\rangle, \{a, b, c\};\\
\quad vis!(tr, ref) \rightarrow tr := tr ^\frown \langle\varepsilon\rangle;\ n, ref := n_1, \{c, \varepsilon\};\\
\quad \sqcap\, e : \{a, b\} \bullet\ vis!(tr, ref) \rightarrow tr := tr ^\frown \langle e\rangle;\ n, ref := n_2, \{a, b, \varepsilon\};\\
\qquad\qquad\qquad\quad vis!(tr, ref) \rightarrow tr := tr ^\frown \langle c\rangle;\ n, ref := n_3, \{a, b, c, \varepsilon\};\\
\qquad\qquad\qquad\quad vis!(tr, ref) \rightarrow Skip
\end{array}$$

□

Besides the healthiness conditions of reactive processes, as defined in Section 2, the processes of interest here satisfy the healthiness condition below.

$$\mathsf{R4}(P) \mathrel{\widehat=} P \wedge \operatorname{ran} tr' \subseteq \{\!| vis |\!\}$$

It ensures that all events observed in the trace are communications over the channel vis. Together with R1, R4 guarantees that this holds for tr and tr'. We use $\operatorname{ran} s$ to denote the set of elements in a sequence s.

The function $\nu(G)$ defined below maps a graph G to a reactive process. It provides an abstract specification for $AS(G)$ using the observational variables of the reactive process theory, rather than programming constructs.

Definition 2. $\nu(G) \mathrel{\widehat=} \mathsf{R}(true \vdash \nu_P(G))$ *where*

$$\begin{array}{l}
\nu_P(G) \mathrel{\widehat=} tr < tr' \Rightarrow\\
\left(\begin{array}{l}
\exists\, tr_M, ref_M \bullet (tr_M, ref_M) = \mathrm{msg} \circ \mathrm{last}\,(tr') \wedge\\
\quad \left(\begin{array}{l}
(\exists\, e, n', r' \bullet G[node(G)(\langle\rangle)/n])\\
\quad \lhd\ tr_M = \langle\rangle\ \rhd\\
\left(\begin{array}{l}
\exists\, r' \bullet ref_M \in r' \wedge\\
\ G[node(G)(\mathrm{front}\ tr_M), \mathrm{last}\ tr_M, node(G)(tr_M)/n, e, n']
\end{array}\right)
\end{array}\right)
\end{array}\right)
\end{array}$$

We use a node-labelling partial function $node(G)$ that maps traces to nodes of

G. It is well defined, because an essential property of a normalised graph is that, for every trace, there is a unique node to which it leads [19, p.161]. We define $\nu(G)$ as a design that specifies that it never diverges: the precondition is *true*. The postcondition $\nu_P(G)$ defines that if an event has occurred ($tr < tr'$), then the behaviour is given by the failure (tr_M, ref_M) communicated in the last event recorded in tr'. We use the function $msg(vis.(tr_M, ref_M)) \mathrel{\widehat{=}} (tr_M, ref_M)$. With tr_M and ref_M, $\nu_P(G)$ specifies that what happens next depends on whether the failure emitted contains the empty trace ($tr_M = \langle\rangle$). If it does, then G has to have a node reachable via $\langle\rangle$. Otherwise, the last two elements of the trace must describe a transition in G. The target of this transition has a set of refusal sets r'; the refusal set in the failure must be an element of r'.

The function $\nu(G)$ is the left (upper) adjoint of a Galois connection between the theories of normalised graphs and reactive processes. To establish this result, and others in the sequel, we use the relationship between an R2-healthy assertion ψ used as a postcondition and the process $Proc(\psi)$ that implements ψ. We define $Proc(\psi) \mathrel{\widehat{=}} \mathsf{R}(true \vdash \psi)$, as a reactive design that requires that the process does not diverge and establishes ψ. Moreover, for a reactive design P, we define a simple way to extract its postcondition $Post(P) \mathrel{\widehat{=}} P[true, true, false/ok, ok', wait]$.

Theorem 1. *The pair* $(Proc, Post)$ *is a Galois connection.*

Proof A design $\phi \vdash \psi$ is defined by $ok \wedge \phi \Rightarrow ok' \wedge \psi$. So, in $\mathsf{R}(\phi \vdash \psi)$, the values of ok and ok' in ψ are defined by the design to be *true*. Moreover, the value of *wait* is defined by R3 to be *false*. Therefore, below we consider, without loss of generality, that ψ does not have free occurrences of ok, ok', or *wait*.

$$
\begin{aligned}
& Post \circ Proc(\psi) \\
&= (\mathsf{R}(true \vdash \psi))[true, true, false/ok, ok', wait] && \text{[definitions of } Post \text{ and } Proc\text{]} \\
&= \mathsf{R1} \circ \mathsf{R2}((true \vdash \psi)[true, true, false/ok, ok', wait]) && \text{[definition of R]} \\
&= \mathsf{R1} \circ \mathsf{R2}(\psi[true, true, false/ok, ok', wait]) && \text{[substitution in a design]} \\
&= \mathsf{R1} \circ \mathsf{R2}(\psi) && [ok,\ ok', \text{ and } wait \text{ are not free in } \psi] \\
&= \mathsf{R1}(\psi) && [\psi \text{ is R2}] \\
&\Rightarrow \psi && \text{[definition of R1 and predicate calculus]}
\end{aligned}
$$

Next, we prove that $Proc \circ Post(P) = I\!I$, so we have a co-retract. We use P^b_a to stand for the substitution $P[a, b/wait, ok']$, and use t and f for *true* and *false*.

$$
\begin{aligned}
& Proc \circ Post(P) \\
&= \mathsf{R}(true \vdash P[true, true, false/ok, ok', wait]) && \text{[definitions of } Proc \text{ and } Post\text{]} \\
&= \mathsf{R}(true \vdash (\mathsf{R}(\neg P^f_f \vdash P^t_f)[true, true, false/ok, ok', wait])) \\
& && \text{[reactive-design theorem: } P = R(\neg P^f_f \vdash P^t_f)\text{]} \\
&= \mathsf{R}(true \vdash \mathsf{R}(\neg P^f_f \Rightarrow P^t_f)) && \text{[substitution]} \\
&= \mathsf{R}(true \vdash \mathsf{R2}(\neg P^f_f \Rightarrow P^t_f)) \\
& && [\mathsf{R} = \mathsf{R1} \circ \mathsf{R2} \circ \mathsf{R3} \text{ and } \mathsf{R1} \circ \mathsf{R3}(P \vdash \mathsf{R1} \circ \mathsf{R3}(P)) = \mathsf{R1} \circ \mathsf{R3}(P \vdash Q)]
\end{aligned}
$$

$$
\begin{aligned}
&= \mathsf{R}(\mathit{true} \vdash \neg P^f_f \Rightarrow P^t_f) && [\text{assumption: } P \text{ is } \mathsf{R2}]\\
&= \mathsf{R}(\neg P^f_f \vdash P^t_f) && [\text{property of a design}]\\
&= P && [\text{reactive-design theorem}]
\end{aligned}
$$

□

For graphs and reactive processes we have the following result.

Theorem 2. *$\nu(G)$ defines a Galois connection.*

Proof From the definition of $\nu(G)$, we have that $\nu(G) = \mathit{Proc} \circ \nu_P(G)$. Since $\nu_P(G)$ is monotonic and universally disjunctive, it defines a Galois connection between normalised graphs and ($\mathsf{R2}$-healthy) assertions. By Theorem 1, *Proc* defines a Galois connection between assertions and reactive processes. The composition of Galois connections is a Galois connection itself. □

With the above theorem, we formalise the point (2) described in Section 1.

4 A UTP theory for Kripke structures

A Kripke structure $(S, s_0, R : \mathbb{P}(S \times S), L : S \to \mathbb{P}\, AP, AP)$ is a quintuple, where S is the set of states, s_0 is the initial state, R is a transition relation between states, and L is a labelling function for states. The labels are sets of atomic propositions from AP that are satisfied by the states. R is required to be total, so that there are no stuck states in a Kripke structure.

In our theory, states are identified with the valuations of variables v, which define the properties satisfied by the states, and so define L and AP. Moreover, we include $pc, pc' : 0 \,..\, 2$ to record a program counter. The value of pc in a state defines whether it is initial, $pc = 0$, intermediate, $pc = 1$, or final, $pc = 2$. Satisfaction of properties is checked in the intermediate states.

In Kripke structures for reactive processes, the other variables of interest are $t_k : \operatorname{seq} \Sigma^\varepsilon$, whose value is the trace performed so far, and $\mathit{ref}_k : \mathbb{P}\, \Sigma^\varepsilon$, whose value is the current refusal, and their dashed counterparts t'_k and ref'_k. We present, however, a theory that is not specific to these variables.

Example 3. Figure 3 gives the Kripke structure for the process in Example 1 and corresponding program in Example 2. In Figure 3, we give the values of the variables t_k and ref_k in each state as a pair. For the states in which $pc = 0$ or $pc = 2$, however, the values of these variables is arbitrary and not given. The states for which $pc = 2$ have self-transitions to avoid stuck states. □

Example 4. The relation EK for the Kripke structure in Figure 3 is as follows.

$$
\begin{aligned}
&pc = 0 \land pc' = 1 \land t'_k = \langle\,\rangle \land \mathit{ref}'_k = \Sigma \lor\\
&pc = pc' = 1 \land t_k = \langle\,\rangle \land \mathit{ref}_k = \Sigma \land t'_k \in \{\langle a\rangle, \langle b\rangle\} \land \mathit{ref}'_k = \{a, b, \varepsilon\} \lor\\
&pc = pc' = 1 \land t_k = \langle a\rangle \land \mathit{ref}_k = \{a, b, \varepsilon\} \land t'_k = \langle a, c\rangle \land \mathit{ref}'_k = \{a, b, c, \varepsilon\} \lor\\
&pc = pc' = 1 \land t_k = \langle b\rangle \land \mathit{ref}_k = \{a, b, \varepsilon\} \land t'_k = \langle b, c\rangle \land \mathit{ref}'_k = \{a, b, c, \varepsilon\} \lor\\
&pc = 1 \land t_k \in \{\langle a, c\rangle, \langle b, c\rangle\} \land pc' = 2 \lor\\
&pc = 2 \land pc' = 2
\end{aligned}
$$

□

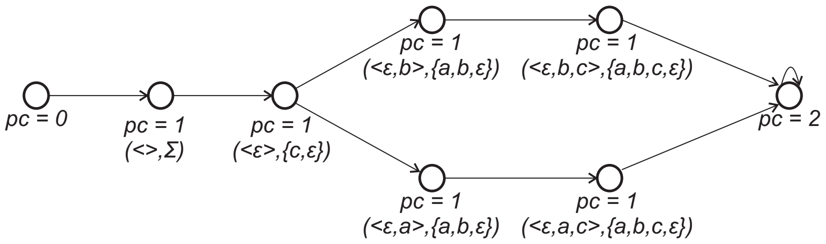

Fig. 3. Kripke structure for $a \rightarrow c \rightarrow STOP \ \Box \ b \rightarrow c \rightarrow STOP$

$\mathsf{HK1}(K) \mathrel{\widehat=} K \land ValT$
$\quad ValT \mathrel{\widehat=} pc = 0 \land pc' = 1 \lor pc = 1 \land pc' \neq 0 \lor pc = 2 \land pc' = 2$
$\mathsf{HK2}(K) \mathrel{\widehat=} SelfT;\ K$
$\quad SelfT \mathrel{\widehat=} pc = pc' \land (pc = 1 \Rightarrow v = v')$
$\mathsf{HK3}(K) \mathrel{\widehat=} K;\ SelfT$
$\mathsf{HK4}(K) \mathrel{\widehat=} K \land \exists\, v, v' \bullet K[0/pc]$

Table 3. Healthiness conditions of the Kripke-structure theory

Healthiness conditions Table 3 presents the healthiness conditions. From the initial state, we move to an intermediate state, and there is no transition back to the initial state or out of the final state. All this is ensured by HK1. With HK2 we establish that the value of v when $pc = 0$ or $pc = 2$ is arbitrary. Similarly, with HK3 we establish that the value of v' when $pc' = 2$ is arbitrary. $SelfT$ specifies transitions that keep the value of pc, but that preserve v only in intermediate states. These are a kind of self transitions. We use $v = v'$ to refer to the conjunction $v_1 = v_1' \land \ldots \land v_n = v_n'$ including an equality for each variable in v and v'. Finally, HK4 requires that either the Kripke structure is empty, or there is a transition from the initial state.

HK1 is conjunctive and so idempotent, and monotonic since $ValT$ does not depend on K. For HK2 and HK3, monotonicity follows from monotonicity of sequence. Idempotence follows from the result below [12, p.90].

Lemma 1. *$SelfT;\ SelfT = SelfT$ where $out\alpha(SelfT) = in\alpha(SelfT) = \{v', pc'\}$.*

The proof of this results and others omitted below can be found in [6].
HK4 is conjunctive and so idempotent, and monotonic since $\exists\, v, v' \bullet K[0/pc]$ is monotonic on K. Commutativity of HK1 with HK2 and HK3 is proved in [6].
Commutativity of HK1 and HK4 is simple because they are both conjunctive.
Commutativity of HK2 and HK3 is established in [12, p.90]. Finally, commutativity of HK2 and HK4, and of HK3 and HK4 are established in [6].
We cannot introduce a healthiness condition $\mathsf{HK}(K) = (true;\ K) \Rightarrow K$ that requires that a Kripke structure is not empty; (like H4 in the case of designs) it is not monotonic. So, we keep *false* in the lattice; it represents miracle, as usual.

Connection from reactive processes As mentioned above, for modelling processes, the additional variables are t_k and ref_k, and their dashed counterparts t_k' and

ref'_k. In this more specific setting, we have the extra healthiness condition below.

$$\begin{array}{l} \mathsf{HK5}(K) \mathrel{\widehat=} K \land \mathit{ValRT} \\ \mathit{ValRT} \mathrel{\widehat=} t'_k = \langle\rangle \land pc = 0 \land pc' = 1 \lor \\ \qquad\qquad t'_k \neq \langle\rangle \land pc = pc' = 1 \land t_k = \mathrm{front}\ t'_k \lor \\ \qquad\qquad pc' = 2 \end{array}$$

The property ValRT defines valid reactive transitions. From the initial state, we reach just the empty trace, and each transition between intermediate states capture the occurrence of a single event: the last event in t'_k.

As discussed previously, we can represent a CSP process G by a reactive process P that outputs in a channel vis the failures of G with maximal refusals. In other words, the events of P define the failures of G. Below, we define how, given a reactive process P whose events are all communications on vis, we construct a corresponding Kripke structure $\kappa(P)$ whose states record the failures of G. To model the state of the action system before it produces any traces or maximal refusals, we let go of HK1 and allow transitions from states for which $pc = 2$ back to an intermediate state with $pc' = 1$.

Definition 3. *$\kappa(P) \mathrel{\widehat=} \kappa_I \circ Post(P)$ where*

$$\kappa_I(P) \mathrel{\widehat=} \exists\, wait', tr, tr', ref, ref' \bullet P \land I_\kappa$$

$$I_\kappa \mathrel{\widehat=} \left(\begin{array}{l} tr = tr' \land pc = 2 \land pc' = 1 \lor \\ \#(tr' - tr) = 1 \land \\ \quad pc = 0 \land pc' = 1 \land (t'_k, ref'_k) = \mathrm{msg} \circ \mathrm{last}\,(tr') \lor \\ \#(tr' - tr) > 1 \land \\ \quad pc = 1 \land pc' = 1 \land \\ \quad (t_k, ref_k) = \mathrm{msg} \circ \mathrm{last}\ \circ \mathrm{front}\,(tr') \land (t'_k, ref'_k) = \mathrm{msg} \circ \mathrm{last}\,(tr') \lor \\ \neg\ wait' \land \#(tr' - tr) > 1 \land \\ \quad pc = 1 \land pc' = 2 \land (t_k, ref_k) = \mathrm{msg} \circ \mathrm{last}\ \circ \mathrm{front}\,(tr') \lor \\ \neg\ wait' \land pc = 2 \land pc' = 2 \end{array}\right)$$

In defining $\kappa(P)$, of interest is the behaviour of P when the previous process did not diverge (ok' is $true$) and terminated ($wait$ is $false$) and P has not diverged (ok' is $true$). This is the postcondition of P, as defined by $Post$. The postcondition has in its alphabet the variables $wait'$, tr, tr', ref, and ref'. In defining κ_I, these variables are quantified, and used in I_κ to define the values of pc, t_k, ref_k, pc', t'_k, and ref'_k from the theory of Kripke structures.

If only one output has occurred ($\#(tr' - tr) = 1$), then the event $vis.(t'_k, ref'_k)$ observed defines the state that can be reached from the initial state of the Kripke structure. When more events have occurred, we define a transition between the intermediate states characterised by the last two events. Prefix closure of P ensures that we get a transition for every pair of events. When P terminates ($\neg\ wait'$), we get two transitions, one from the last event to the final state ($pc' = 2$), and the loop transition for the final state.

To establish that $\kappa(P)$ defines a Galois connection, we use a result in [12] proved in [6]. It considers functions L and R between lattices A and B (ordered by $\sqsubseteq$) with alphabets a and c, when L and R are defined in terms of a predicate I over the alphabet defined by the union of a and c. We can see these functions as establishing a data refinement between A and B with coupling invariant I.

Theorem 3. *L and R defined below are a Galois connection between A and B.*

$$L(P_C) \mathrel{\widehat=} \exists\, c \bullet P_C \wedge I \quad \text{and} \quad R(P_A) \mathrel{\widehat=} \forall\, a \bullet I \Rightarrow P_A$$

This result can be used to prove the following theorem.

Theorem 4. *$\kappa(P)$ defines a Galois connection.*

Proof From Theorem 1, we know that *Post* defines a Galois connection. Theorem 3 establishes that κ_I defines a Galois connection as well. Their composition, which defines κ, therefore, also defines a Galois connection. □

The above theorem formalises the point (3) mentioned in Section 1.

5 A UTP theory for traces and maximal refusals

This is a theory of conditions (predicates on a single state) with alphabet ok_M, tr_M and ref_M. These variables are similar to those of the theory of reactive processes, but ref_M records only maximal refusals. We use the notion of refinement in this theory to define satisfaction for relations in all our theories.

As can be expected, there is a rather direct Galois connection between reactive processes and definitions of traces and maximal refusals in this theory.

Definition 4. $\theta(P) \mathrel{\widehat=} \theta_P \circ Post(P)$ *where*

$$\begin{array}{l}\theta_P(P) \mathrel{\widehat=} \exists\, wait', tr, tr', ref, ref' \bullet P \wedge I_\theta \\ I_\theta \mathrel{\widehat=}\; ok_M = (tr' > tr) \wedge (ok_M \Rightarrow (tr_M, ref_M) = \mathrm{msg} \circ \mathrm{last}\,(tr'))\end{array}$$

In defining the failures of $\theta(P)$, we need the postcondition of P. From that, we obtain failures once P has started communicating, so ok_M is characterised by $(tr' > tr)$. If we do have a failure, it is that last communicated via *vis* in tr'.

Theorem 5. *$\theta(P)$ defines a Galois connection.*

Proof Similar to that of Theorem 4. □

The healthiness conditions of a theory of traces and refusals are well known [19]. We record, however, via the healthiness condition HM below the role of ok_M, as a flag that indicates whether observations are valid.

$$\mathsf{HM}(M) \mathrel{\widehat=} ok_M \Rightarrow M$$

(This is just the healthiness condition H1 of the UTP theory of designs, which first introduces the use of *ok*). The predicates of our theory of traces and maximal refusals are used as conditions in our satisfaction relations presented next.

5.1 Satisfaction for normalised graphs

The function $\eta(G)$ defines a Galois connection between the theory of normalised graphs and the theory of traces and maximal refusals.

Definition 5. $\eta(G) \mathrel{\widehat{=}} \exists\, n, e, n', r' \bullet ok_M \Rightarrow G \wedge I_\eta$ *where*

$$I_\eta \mathrel{\widehat{=}} \left((n = node(G)(\langle\rangle)) \lhd tr_M = \langle\rangle \rhd \left(\begin{array}{l} n = node(G)(\text{front}\; tr_M) \wedge \\ e = \text{last}\; tr_M \wedge \\ n' = node(G)(tr_M) \wedge \\ ref_M \in r' \end{array} \right) \right)$$

As required, we define $\eta(G)$ by characterising traces tr_M and refusals ref_M using the variables n, e, n' and r' from the theory of graphs. If ok_M is $true$, then tr_M is empty if the current node n can be reached with the empty trace (that is, it is the initial node). Otherwise, the trace is that used to reach n concatenated with $\langle e \rangle$. Moreover, ref_M is a refusal in the label r' of the target node.

To establish that $\eta(G)$ is the left adjoint of a Galois connection between the theories of normalised graphs and of maximal refusals, we use the following general result, similar to that in Theorem 3.

Theorem 6. *L and R defined below are a Galois connection between A and B.*

$$L(P_C) \mathrel{\widehat{=}} \exists\, c \bullet b \Rightarrow P_C \wedge I \quad \text{and} \quad R(P_A) \mathrel{\widehat{=}} \forall\, a \bullet I \Rightarrow P_A$$

where b is a boolean variable in the alphabet a of A, and $\mathsf{HC}(P_A) = b \Rightarrow P_A$ is a healthiness condition of the lattice B.

The proof is similar to that of Theorem 3 and can be found in [6].

Theorem 7. *$\eta(G)$ defines a Galois connection.*

Proof. Direct application of Theorem 6.

Using $\eta(G)$, we can use refinement in the theory of traces and maximal refusals to define satisfaction as shown below.

Definition 6. *For a property ϕ and a graph G, we define G sat $\phi \mathrel{\widehat{=}} \phi \sqsubseteq \eta(G)$.*

Normalised graphs G have the same traces and maximal refusals as the reactive program $\nu(G)$ that it characterises.

Theorem 8. $\eta(G) = \theta \circ \nu(G)$

Proof

$$\begin{array}{ll} \theta \circ \nu(G) & \\ = \exists\, tr, tr' \bullet \nu_P(G) \wedge & [\text{definitions of } \theta \text{ and } \nu] \\ \quad ok_M = (tr' > tr) \wedge (ok_M \Rightarrow (tr_M, ref_M) = \text{msg} \circ \text{last}\,(tr')) & \end{array}$$

$$
\begin{array}{l}
= \exists\, tr, tr' \bullet \hfill \text{[definition of } \nu_P\text{]} \\
\quad \left(\begin{array}{l}
tr < tr' \Rightarrow \\
\quad \left(\begin{array}{l}
\exists\, tr_M, ref_M \bullet (tr_M, ref_M) = \mathrm{msg} \circ \mathrm{last}\,(tr') \land \\
\quad \left(\begin{array}{l}
(\exists\, e, n', r' \bullet G[node(G)(\langle\rangle)/n]) \\
\quad \lhd\ tr_M = \langle\rangle\ \rhd \\
\left(\begin{array}{l}
\exists\, r' \bullet ref_M \in r' \land \\
\quad G[node(G)(\mathrm{front}\ tr_M), \mathrm{last}\ tr_M, node(G)(tr_M) \\
\qquad /n, e, n']
\end{array} \right)
\end{array} \right)
\end{array} \right)
\end{array} \right) \\
\quad \land \\
\quad ok_M = (tr' > tr) \land (ok_M \Rightarrow (tr_M, ref_M) = \mathrm{msg} \circ \mathrm{last}\,(tr'))
\end{array}
$$

$$
\begin{array}{l}
= \neg\, ok_M \lor \hfill \text{[case analysis on } ok_M\text{]} \\
\quad \exists\, tr, tr' \bullet \\
\quad \left(\begin{array}{l}
\exists\, tr_M, ref_M \bullet (tr_M, ref_M) = \mathrm{msg} \circ \mathrm{last}\,(tr') \land \\
\quad \left(\begin{array}{l}
(\exists\, e, n', r' \bullet G[node(G)(\langle\rangle)/n]) \\
\quad \lhd\ tr_M = \langle\rangle\ \rhd \\
\left(\begin{array}{l}
\exists\, r' \bullet ref_M \in r' \land \\
\quad G[node(G)(\mathrm{front}\ tr_M), \mathrm{last}\ tr_M, node(G)(tr_M)/n, e, n']
\end{array} \right)
\end{array} \right)
\end{array} \right) \\
\quad \land \\
\quad (tr' > tr) \land (tr_M, ref_M) = \mathrm{msg} \circ \mathrm{last}\,(tr')
\end{array}
$$

$$
\begin{array}{l}
= \neg\, ok_M \lor \hfill \text{[predicate calculus]} \\
\quad \exists\, tr, tr' \bullet \\
\quad \left(\begin{array}{l}
(\exists\, e, n', r' \bullet G[node(G)(\langle\rangle)/n]) \\
\quad \lhd\ tr_M = \langle\rangle\ \rhd \\
\left(\begin{array}{l}
\exists\, r' \bullet ref_M \in r' \land \\
\quad G[node(G)(\mathrm{front}\ tr_M), \mathrm{last}\ tr_M, node(G)(tr_M)/n, e, n']
\end{array} \right)
\end{array} \right) \\
\quad \land \\
\quad (tr' > tr) \land (tr_M, ref_M) = \mathrm{msg} \circ \mathrm{last}\,(tr')
\end{array}
$$

$$
\begin{array}{l}
= \neg\, ok_M \lor \hfill \text{[predicate calculus]} \\
\left(\begin{array}{l}
(\exists\, e, n', r' \bullet G[node(G)(\langle\rangle)/n]) \\
\quad \lhd\ tr_M = \langle\rangle\ \rhd \\
\left(\begin{array}{l}
\exists\, r' \bullet ref_M \in r' \land \\
\quad G[node(G)(\mathrm{front}\ tr_M), \mathrm{last}\ tr_M, node(G)(tr_M)/n, e, n']
\end{array} \right)
\end{array} \right) \\
\land \\
\exists\, tr, tr' \bullet (tr' > tr) \land (tr_M, ref_M) = \mathrm{msg} \circ \mathrm{last}\,(tr')
\end{array}
$$

$$
\begin{array}{l}
= \neg\, ok_M \lor \hfill \text{[predicate calculus]} \\
\left(\begin{array}{l}
(\exists\, e, n', r' \bullet G[node(G)(\langle\rangle)/n]) \\
\quad \lhd\ tr_M = \langle\rangle\ \rhd \\
\left(\begin{array}{l}
\exists\, r' \bullet ref_M \in r' \land \\
\quad G[node(G)(\mathrm{front}\ tr_M), \mathrm{last}\ tr_M, node(G)(tr_M)/n, e, n']
\end{array} \right)
\end{array} \right)
\end{array}
$$

$$
\begin{array}{l}
= \neg\, ok_M \lor \hfill \text{[predicate calculus]} \\
\left(\begin{array}{l}
(\exists\, n, e, n', r' \bullet G \land n = node(G)(\langle\rangle)) \\
\quad \lhd\ tr_M = \langle\rangle\ \rhd \\
\left(\begin{array}{l}
\exists\, n, e, n, r' \bullet ref_M \in r' \land \\
\quad G \land n = node(G)(\mathrm{front}\ tr_M) \land e = \mathrm{last}\ tr_M \land n' = node(G)(tr_M)
\end{array} \right)
\end{array} \right)
\end{array}
$$

$$= \exists\, n, e, n', r' \bullet ok_M \Rightarrow G \land \left(\begin{array}{l} (n = node(G)(\langle\rangle)) \\ \quad \lhd\, tr_M = \langle\rangle \,\rhd \\ \left(\begin{array}{l} ref_M \in r' \land \\ n = node(G)(\text{front}\, tr_M) \land e = \text{last}\, tr_M \land n' = node(G)(tr_M) \end{array} \right) \end{array} \right) \qquad \text{[property of conditional]}$$

$$= \eta(G) \qquad \text{[definition of } \eta\text{]}$$

□

This establishes that our transformations preserve traces and maximal refusals. So, to check satisfaction for a graph G, we can use $\theta \circ \nu(G)$, instead of $\eta(G)$.

5.2 Satisfaction for Kripke structures

The function $\zeta(G)$ defines a Galois connection between the theory of Kripke structures and the theory of traces and maximal refusals.

Definition 7.

$$\begin{array}{l} \zeta(K) \mathrel{\widehat=} \exists\, pc, pc', t_k, t_k', ref_k, ref_k' \bullet K \land \\ \quad ok_M = (pc \in \{0, 1\}) \land (ok_M \Rightarrow tr_M = t_k' \land ref_M = ref_k') \end{array}$$

The traces tr_M and refusals ref_M that are captured are those of the target states.

Theorem 9. *$\zeta(K)$ defines a Galois connection.*

Proof Direct consequence of Theorem 3. □

Using ζ, we can use refinement in the theory of traces and maximal refusals to define satisfaction for Kripke structures as shown below.

Definition 8. *For a property ϕ and a Kripke structure K, we define*

$$K \text{ sat } (pc' = 1 \Rightarrow \phi) \mathrel{\widehat=} \phi \sqsubseteq \zeta(K \land pc' = 1)$$

The Kripke structures $\zeta \circ \kappa(P)$ have the same traces and maximal refusals as the reactive process P that they characterise.

Theorem 10. $\theta(P) = \zeta(\kappa(P) \land pc' = 1)$

Proof

$$\zeta(\kappa(P) \land pc' = 1)$$

$$= \zeta \left(\begin{array}{l} \exists\, wait', tr, tr', ref, ref' \bullet Post(P) \land \\ \quad \left(\begin{array}{l} \#(tr' - tr) = 1 \land pc = 0 \lor \\ \#(tr' - tr) > 1 \land pc = 1 \land (t_k, ref_k) = \text{msg} \circ \text{last}\,(\text{front}\, tr') \lor \\ tr' = tr \land pc = 2 \end{array} \right) \land \\ \quad pc' = 1 \land (t_k', ref_k') = \text{msg} \circ \text{last}\,(tr') \end{array} \right)$$

[definition of $\kappa(P)$ and predicate calculus]

$$= \left(\begin{array}{l} \exists\, pc, pc', t_k, t_k', ref_k, ref_k' \bullet \\ \quad \left(\begin{array}{l} \exists\, wait', tr, tr', ref, ref' \bullet Post(P) \land \\ \quad \left(\begin{array}{l} tr = tr' \land pc = 2 \lor \\ \#(tr' - tr) = 1 \land pc = 0 \lor \\ \#(tr' - tr) > 1 \land pc = 1 \land (t_k, ref_k) = \mathrm{msg} \circ \mathrm{last}\,(\mathrm{front}\ tr') \end{array} \right) \\ \quad \land \\ \quad pc' = 1 \land (t_k', ref_k') = \mathrm{msg} \circ \mathrm{last}\,(tr') \end{array} \right) \\ \land \\ ok_M = (pc \in \{0, 1\}) \land (ok_M \Rightarrow tr_M = t_k' \land ref_M = ref_k') \end{array} \right)$$

[definition of ζ]

$$= \left(\begin{array}{l} \exists\, wait', tr, tr', ref, ref' \bullet Post(P) \land \\ \quad \exists\, pc, t_k', ref_k' \bullet \\ \qquad \left(\begin{array}{l} tr = tr' \land pc = 2 \lor \\ \#(tr' - tr) = 1 \land pc = 0 \lor \#(tr' - tr) > 1 \land pc = 1 \end{array} \right) \land \\ \qquad (t_k', ref_k') = \mathrm{msg} \circ \mathrm{last}\,(tr') \land \\ \qquad ok_M = (pc \in \{0, 1\}) \land (ok_M \Rightarrow tr_M = t_k' \land ref_M = ref_k') \end{array} \right)$$

[predicate calculus]

$$= \left(\begin{array}{l} \exists\, wait', tr, tr', ref, ref' \bullet Post(P) \land \\ \quad \left(\begin{array}{l} \exists\, pc \bullet \\ \quad tr = tr' \land pc = 2 \lor \\ \quad \#(tr' - tr) = 1 \land pc = 0 \lor \#(tr' - tr) > 1 \land pc = 1 \end{array} \right) \land \\ \quad ok_M = (pc \in \{0, 1\}) \land (ok_M \Rightarrow (t_M, ref_M) = \mathrm{msg} \circ \mathrm{last}\,(tr')) \end{array} \right)$$

[predicate calculus]

$$= \left(\begin{array}{l} \exists\, wait', tr, tr', ref, ref' \bullet Post(P) \land \\ \quad ok_M = tr' > tr \land (ok_M \Rightarrow (t_M, ref_M) = \mathrm{msg} \circ \mathrm{last}\,(tr')) \end{array} \right)$$

[predicate calculus]

$= \theta(P)$ [definition of $\theta(P)$]

□

This establishes the semantic preservation of our transformation.

As a consequence, it is direct that, to check satisfaction for a graph G, we can use $\kappa \circ \nu(G)$, instead of $\eta(G)$, as shown below.

Theorem 11. G sat $\phi \Leftrightarrow \kappa \circ \nu(G)$ sat $(pc' = 1 \Rightarrow \phi)$

Proof

G sat ϕ

$= \phi \sqsubseteq \eta(G)$ [definition of sat]

$= \phi \sqsubseteq \theta \circ \nu(G)$ [Theorem 8]

$= \phi \sqsubseteq \zeta(\kappa \circ \nu(G) \land pc' = 1)$ [Theorem 10]

$= \kappa \circ \nu(G)$ sat $(pc' = 1 \Rightarrow \phi)$ [definition of sat]

□

This is the main result of this paper.

6 Conclusions

We have previously developed a monitor for runtime verification of sequential nondeterministic programs with Kripke-structure semantics. It can check the program's execution behaviour against a subset of LTL safety formulas.

In this paper, we have presented novel UTP theories for normalised graphs and Kripke structures that model these nondeterministic programs. They are complete lattices under the UTP refinement order. Our relation of interest, however, is satisfaction, which we have defined for graphs and Kripke structures. Using this framework, we can justify the soundness of the translation of CSP models via normalised graphs into Kripke structures. This induces a concrete translation from CSP processes to sequential nondeterministic programs.

The framework also indicates how to translate a subset of safety formulas into CSP specifications on traces and refusals. These formulas belong to the formula class handled by the runtime monitor. The framework guarantees that an execution of some CSP process P satisfies a specification a given specification if, and only if, P's translation into a sequential nondeterministic program does.

Temporal model checking of UTP designs (pre and postcondition pairs) based on Kripke structures is discussed in [1]. Like we do, [1] defines satisfaction as an extra relation in a lattice ordered by refinement. Satisfaction is defined for states, and temporal logic operators are modelled as fixed-point operators. We adopt a similar notion of state as variable valuations, but do not formalise temporal operators. On the other hand, we define explicitly a theory of Kripke structures, rather than encode them as designs. Moreover, we capture the relationship between Kripke structures and failure models: directly to action systems encoded as reactive processes and indirectly to normalised graphs. As far as we know, we give here the first account of automata-based theories in the UTP.

An issue we have not covered is the relationship of our theories with the existing UTP CSP theory [12, 7]. That amounts to formalising the operational semantics of CSP and the normalisation algorithm of FDR3. Since maximal refusals cannot be deduced from the denotational semantics of CSP [19, p.124], we do not expect an isomorphism between the theories.

An important property of normalised graphs and Kripke structures that is not captured by our healthiness conditions is connectivity. The definition of a monotonic idempotent that captures this property is left as future work. For Kripke structures, we also do not capture the fact that there are no intermediate stuck states. If we consider that every assignment of values to v is a valid state, then this can be captured by the function $\mathsf{HK6}(K) = (K;\ \mathit{true}) \Rightarrow K$.

Acknowledgements The work of Ana Cavalcanti and Jim Woodcock is funded by the EPSRC grant EP/H017461/1 and the EU INTO-CPS. No new primary data were created during this study. The work of Wen-ling Huang and Jan Peleska is funded by the grant *ITTCPS – Implementable Testing Theory for Cyber-physical Systems*as part of the German Universities Excellence Initiative.

References

1. Anderson, H., Ciobanu, G., Freitas, L.: UTP and Temporal Logic Model Checking. In:Unifying Theories of Programming, LNCS, vol. 5713, pp. 22–41. Springer (2010)
2. Back, R.J., Kurki-Suonio, R.: Distributed cooperation with action systems. ACM Transactions on Programming Languasge Systems 10(4), 513–554 (1988)
3. Burdy, L.et al: An overview of JML tools and applications. STTT 7(3), 212 – 232 (2005)
4. Butterfield, A.: A denotational semantics for handel-c. FACJ 23(2), 153–170 (2011)
5. Cavalcanti, A.L.C., Clayton, P., O'Halloran, C.: From Control Law Diagrams to Ada via *Circus*. FACJ 23(4), 465–512 (2011)
6. Cavalcanti, A.L.C., Huang, W.L., Peleska, J., Woodcock, J.C.P.: Unified Runtime Verification for CSP - Extended version. Tech. rep., University of York, Department of Computer Science, York, UK (2015), available at `www.cs.york.ac.uk/circus/hijac/publication.html`
7. Cavalcanti, A.L.C., Woodcock, J.C.P.: A Tutorial Introduction to CSP in Unifying Theories of Programming. In: Refinement Techniques in Software Engineering. LNCS, vol. 3167, pp. 220–268. Springer (2006)
8. Cavalcanti, A.L.C., Zeyda, F., Wellings, A., Woodcock, J.C.P., Wei, K.: Safety-critical Java programs from *Circus* models. RTS 49(5), 614–667 (2013)
9. Gibson-Robinson, T., Armstrong, P., Boulgakov, A., Roscoe, A.W.: FDR3 A Modern Refinement Checker for CSP. In: TACAS. pp. 187–201 (2014)
10. Haddad, G., Hussain, F., Leavens, G.T.: The Design of SafeJML, A Specification Language for SCJ with Support for WCET Specification. In: JTRES. ACM (2010)
11. Harwood, W., Cavalcanti, A.L.C., Woodcock, J.C.P.: A Theory of Pointers for the UTP. In: ICTAC. LNCS, vol. 5160, pp. 141–155. Springer (2008)
12. Hoare, C.A.R., Jifeng, H.: Unifying Theories of Programming. Prentice-Hall (1998)
13. Huang, W.L., Peleska, J., Schulze, U.: Contract Support for Evolving SoS. Public Document D34.3, COMPASS (2014)
14. Liu, Z., Jifeng, H., Li, X.: rCOS: Refinement of Component and Object Systems. In: FMCO. LNCS, vol. 3657. Springer-Verlag (1994)
15. Miyazawa, A., Lima, L., Cavalcanti, A.L.C.: Formal Models of SysML Blocks. In: ICFEM. LNCS, vol. 8144, pp. 249–264. Springer (2013)
16. Oliveira, M.V.M., Cavalcanti, A.L.C., Woodcock, J.C.P.: A UTP Semantics for *Circus*. FACJ 21(1-2), 3–32 (2009)
17. Peleska, J.: Translating Testing Theories for Concurrent Systems. In: Correct System Design, Essays Dedicated to Ernst-Rüdiger Olderog on the Occasion of his 60th Birthday. LNCS, Springer (2015)
18. Roscoe, A.W. (ed.): A Classical Mind: Essays in Honour of C. A. R. Hoare. Prentice Hall International (UK) Ltd. (1994)
19. Roscoe, A.W.: Understanding Concurrent Systems. Texts in Computer Science, Springer (2011)
20. Sherif, A., Cavalcanti, A.L.C., He, J., Sampaio, A.C.A.: A process algebraic framework for specification and validation of real-time systems. FACJ 22(2), 153–191 (2010)
21. Zeyda, F., Santos, T.L.V.L., Cavalcanti, A.L.C., Sampaio, A.C.A.: A modular theory of object orientation in higher-order UTP. In: FM. LNCS, vol. 8442, pp. 627–642. Springer (2014)
22. Zhu, H., He, J., Qin, S., Brooke, P.: Denotational semantics and its algebraic derivation for an event-driven system-level language. FACJ 27(1), 133–166 (2015)