A Design by Contract Approach to Recover the Architectural Style from Run-time Misbehaviour

Kyriakos Poyias, Emilio Tuosto

University of Leicester, UK

Abstract

We propose to control the reconfigurations of applications leading to an erroneous state by exploiting its architectural model. Our work relies on Architectural Design Rewriting (ADR, for short) which is a rule-based formal framework for modelling (the evolution of) software architectures. We equip the reconfiguration rules of an ADR architecture with pre- and post-conditions expressed in a simple logic; a pre-condition constrains the applicability of a rule while a post-condition specifies the properties expected of the resulting graphs. We give an algorithm to compute the weakest pre-condition out of a rule and its post-condition. On top of this algorithm, we design a simple methodology that allows us to select which rules can be applied at the architectural level to reconfigure a system so to regain its architectural style when it becomes compromised by unexpected run-time reconfigurations.

Keywords: Design By Contract, Software Architectures, Architectural Style

1. Introduction

Modern applications are very rarely developed as “stand-alone” software; as a matter of fact, even simple applications are nowadays open in the sense that they are typically able to connect and/or be integrated with other applications such as those in service-oriented or cloud computing. Interestingly, this kind of software seems to intrinsically call for autonomic features. In fact, it is necessary to guarantee some degree of self-adaptation of applications that can be dynamically composed so that they can automatically adapt to the (often unpredictable) run-time changes. We will not make an attempt to define the concept of autonomic or self-adaptive software. Intuitively, autonomic systems can be thought of as those systems that can adapt response to dynamic changes of the (physical and/or execution) environment into which they act. This intuition is hard to transfer to software (architectures) as argued in [1]. For our purposes
it is enough to consider applications that may run into an array of unexpected states which is so wide to make it impractical a complete specification [? ].

Openness magnifies the complexity of such software. In fact, open applications are subject to unexpected reconfigurations that may hinder their execution and drive computations into erroneous states in an unanticipated manner. Not only it is crucial to detect those states of the computation, but it is necessary to design applications able to re-establish correct configurations in presence of misbehaviour. For example, the reaction to the failure of a service $S$, may redirect the requests of the clients to another service $S'$. Our take is that a practical approach for the development of this kind of software is that the designer specifies the 'normal' behaviour and then (s)he tries to enumerate the (relatively few) erroneous states (that are most likely to happen and for which plausible reconfigurations can be designed). It is desirable to have mechanisms to support designers in this phase. For instance, having executable models or abstract representations of the computations may provide a rich framework into which applications can be probed under different conditions. More importantly, one would benefit from a framework capable to suggest possible ways to bring executions back to an acceptable state (which is of course dependent on the application).

Another problem that can arise in those cases is that the run-time reconfigurations may compromise the alignment with the expected abstract architecture. In the client-service scenario mentioned above, the choice of $S'$ may cause the violation of some architectural constraints designed e.g. to balance the load. Software architectures specify the structure and interconnections of applications. Ordinary computation can change the state, but they are very rarely allowed to modify the architectural style. In this context it is also crucial to preserve architectural styles [2] that allow one (i) to specify (reusable) design patterns, (ii) to confine the parts to be reconfigured, and (iii) to control the architectural changes.

In this paper we propose a solution to the problems above that uses high-level designs of software architectures to automatically compute reconfigurations that drive restore desirable architectural properties (expressed as logical invariants). Roughly, we envisage architectural styles according to the equation:

$$\text{architectural style} = \text{reconfiguration rules} + \text{invariants}$$

where an invariant is the property the designer requires of the application. More precisely, we consider asserted (reconfiguration) rules and use them as contracts that guarantee properties when the rules are applied. When such contracts are violated by the run-time configuration, our framework suggests to designers possible reconfigurations to recover the execution.

We remark that the designer intervention is climacteric as automatic reconfigurations may lead the application into non acceptable states. Therefore, we argue that it is necessary that the automatically computed reconfigurations are vetted by the designer.
A summary of our contributions. We propose a design-by-contract (DbC) approach to tackle the problems discussed above. Our main contribution is an algorithm that computes a weakest pre-condition \( \psi \) out of a post-condition \( \varphi \) and a reconfiguration rule. We prove a theorem that guarantees that the application of the rule to a configuration satisfying \( \psi \) yields a configuration satisfying the post-condition \( \varphi \). This algorithm can be used to compute a reconfiguration if the current configuration violates the invariant.

Our approach hinges on a formal language for specifying software architectures, their refinements, and their style. Methodologically, we adopt ADR [3] as our architectural description language. As surveyed in §2, ADR models systems as (hyper)graphs that is a set of (hyper)edges sharing some nodes; respectively, edges represent distributed components (at some level of abstraction) while nodes represent communication ports. Also, ADR features refinement rules of the form \( L \rightarrow R \) where \( L \) is a (hyper)edge and \( R \) a (hyper)graph meant to replace \( L \) with \( R \) within a given graph. In ADR, a system corresponds to a configuration of elements (i.e. nodes and edges) that can be related to the architecture graph components and expected to respect the architectural style specified by the refinement rules. Such elements can interact through their connections according to run-time interactions (run-time reconfigurations) not represented at the architectural level. A main reason for adopting ADR is that it has been designed to support the alignment of architecture-related information with run-time behaviour in order to drive execution.

A technical contribution of this paper (§3 and §4) is to generalise ADR with asserted productions, that is refinement rules of the form

\[
\{ \psi \} L \rightarrow R \{ \varphi \}
\]  

where the intuition of (2) is that the replacement of \( L \) with \( R \) in a graph that satisfy the pre-condition \( \psi \), yields a graph satisfying the post-condition \( \varphi \). In ADR, architectural styles are formalised in terms of productions that describe the legal configurations of systems. We generalise this by envisaging architectural styles as set of productions together with invariants (expressed as formulae of our logic) which can be thought of as contracts that architectures have to abide by. Pre- and post-conditions are expressed in a simple logic for hypergraphs.

We use asserted production and the algorithm to distil weakest pre-conditions on them (given in §5) to model the semantic aspects of the Java remote method invocation mechanism.

Another contribution exploits equation (1) and the algorithm. Also, we devise an improved methodology to re-establish the architectural style specified for a system when run-time reconfigurations compromise it.

Synopsis. A short overview of ADR is given in §2 (for simplicity, we do not describe ADR reconfiguration; the interested reader is referred e.g. to [3] for the technical details). We introduce a simple logic for ADR in §3. Basic definitions to specify our algorithm are in §4 while the algorithm is in §5. §6 illustrates
our approach on how asserted productions can guarantee design properties by considering a simplified version of Java’s remote method invocation (JRMI).

In § 7 we describe a methodology that relies on the algorithm in § 5 to recover architectural styles compromised by run-time reconfigurations. An application of the methodology is given in § 8. Related work are discussed in § 9. Concluding remarks and future work is in § 10.

Remark. The main ideas of this paper have been introduced in our paper [? ]. Here, we added a model of the semantics of the Java remote method invocation and the proofs of the correctness of our algorithm. In particular, we have taken advantage of the space allowed and refined the definition for computing the weakest pre-conditions (cf. Definition 12) which also relies on a simplified auxiliary map (cf. Definition 11). Also, the methodology in § 7 is an improvement of the one presented in [? ] due to the fact that it recursively computes the weakest pre-condition computed by the algorithm until a possible sequence of reconfigurations is identified to re-establish the architectural style of the system.

2. A walk through ADR

We briefly overview ADR; we borrow from [3] the main definitions and notations (slightly adapting them to our needs).

In the following, $\mathcal{N}$ and $\mathcal{E}$ are two countably infinite and disjoint sets (of nodes and edges respectively), $X^* \overset{\text{def}}{=} \{(x_1, \ldots, x_n) \mid x_1, \ldots, x_n \in X\}$ is the set of finite lists on a set $X$, and $\tilde{x}$ ranges over $X^*$. Also, abusing notation, we sometimes use $\tilde{x}$ to indicate its underlying set of elements.

**Definition 1 ((Hyper)graphs).** A (hyper)graph is a tuple $G = \langle V, E, t \rangle$ where $V \subseteq \mathcal{N}$ and $E \subseteq \mathcal{E}$ are finite and $t : E \to V^*$ is the tentacle function.

Given a graph $G$, we denote with $V_G$, $E_G$, and $t_G$ its nodes, edges, and tentacle function, respectively. An edge $e \in E_G$ is connected to a list of nodes via $t_G$ and the arity of $e$ is the length of $t_G(e)$. It is convenient to write $e(\tilde{u}) \in G$ for $e \in E_G$, $t_G(e) = \tilde{u} \subseteq V_G$.

**Definition 2 (Graph morphism).** Let $G$ and $H$ be two graphs. A graph morphism from $G$ to $H$ is a pair of functions $\langle \sigma_V : V_G \to V_H, \sigma_E : E_G \to E_H \rangle$ s.t. $\sigma_V$ and $\sigma_E$ preserve the tentacle functions, i.e. $\sigma_V \circ t_G = t_H \circ \sigma_E$, where $\sigma_V^* \overset{\text{def}}{=} \text{the homomorphic extension of } \sigma_V$ to $V^*_G$.

In ADR, graphs are typed over a fixed type graph via typing morphisms. As usual an ADR graph $G$ is typed over a type graph $\Gamma$ through $\tau_G$ if $\tau_G$ is a morphism from $G$ to $\Gamma$.

**Definition 3 (ADR graph).** Let $\Gamma$ be a type graph equipped with a map $\eta : E_\Gamma \to \{0, 1\}$. An ADR graph $G$ is a (hyper)graph typed over $\Gamma$ through $\tau_G$ if $\tau_G$ is a morphism from $G$ to $\Gamma$; we call $e \in E_G$ terminal if $\eta(\sigma(e)) = 0$ and non-terminal if $\eta(\sigma(e)) = 1$. 

This is reminiscent of string grammars where terminal symbols correspond to terminal edges and non-terminal symbols to non-terminal edges.

**Example 1.** Let $V = \{\bullet\} \subseteq N$ and $E = \{C, FF, Fls, Fl, BF, P, PF\} \subseteq E$. Consider the type graph $\Gamma = (V, E, t, \eta)$ where $t : C \mapsto (\bullet)$ and $t : e \mapsto (\bullet, \bullet)$ for each $e \in E \setminus \{C\}$, with $\eta(e) = 0$ if $e \in \{C, FF\}$ and $\eta(e) = 1$ otherwise. The graph $G = \langle \{u_1, u_2, u_3, u_4\}, \{ff, fl_1, fl_2\}, t' \rangle$ where $t'$ is defined as $ff \mapsto (u_2, u_1)$, $fl_1 \mapsto (u_3, u_2)$, and $fl_2 \mapsto (u_4, u_2)$ can be typed on $\Gamma$ by $\tau_G$ mapping all the nodes to $\bullet$, $fl_1$ and $fl_2$ to $Fls$, and $ff$ to $FF$.

Hereafter, we fix a typed graph $\Gamma$ and tacitly assume that all graphs $G$ are typed over $\Gamma$ via a morphism $\tau_G$. Intuitively, $\Gamma$ yields the vocabulary of the architectural elements to be used in the designs; moreover, $\Gamma$ specifies how these elements can be connected together (e.g., as in Example 1).

Type and typed graphs have a convenient visual notation. Nodes are circles and edges are drawn as (labelled) boxes; single- and double-lined boxes represent terminal and non-terminal edges, respectively. Tentacles are depicted as lines connecting boxes to circles; conventionally, directed tentacles indicate the first node attached to the edge and the others are taken clockwise. The visual notation for typed graphs include the graph and its typing morphism. Nodes are paired with their types while an edge label $e : e'$ represents the fact that the typing morphism maps the edge $e$ of the graph to the edge $e'$ of the type graph.

**Example 2.** In the visual notation described above, the type graph $\Gamma$ and the graph $G$ of Example 1 can be respectively drawn as

```
FF -> (bullet)
C
```

```
ff : FF -> (u2, u1)
|   |
|   |
fl_1 : Fls -> (u3, u2)
```

where, to simplify the type graph, we use $e \in E \setminus \{C, FF\}$ (instead of drawing an edge for each non-terminal edge of $\Gamma$).

**Definition 4 (Typed Graph morphisms).** A morphism between $\Gamma$-typed graphs $f : G_1 \rightarrow G_2$ is a typed graph morphism if it preserves the typing, i.e. such that $\tau_{G_1} = \tau_{G_2} \circ f$.

**Definition 5 (Productions).** A (design) production $p$ is a tuple $\langle L, R, i : V_L \rightarrow V_R \rangle$ where $L$ is a graph consisting only of a non-terminal edge attached to distinct nodes; $R$ is an ADR graph (with both terminal and non-terminal edges); the nodes in $Im(i)$ (the image of $i$) are called interface nodes.

Design productions can be thought of as rewriting rules that, when applied to a graph $G$, replace a non-terminal (hyper)edge of $G$ matching $L$ with a fresh copy of $R$ (we remark that our morphisms are type-preserving). Also productions have a suitable visual representation illustrated in the next example.
Example 3. The graphical representation below represents a design production.

Since the production above will be used later (cf. Example 7) we will refer to it as bookFlight. The left-hand-side (LHS) of bookFlight is an edge of type Fls (denoted in the left-upper corner of the dotted-box) whose nodes are those outside the dotted box; we omit the identities of such nodes when immaterial. The right-hand-side (RHS) of bookFlight is the graph inside the dotted box. The mapping i of bookFlight is represented by the dotted lines.

The application of asserted productions (cf. Definition 9) encompasses that of ADR productions hence we give here only an example to illustrate how productions are applied.


Note that the rest of the graph (consisting only of the edge ff) including the interface nodes is left unchanged while a fresh node $u_2$ is created.

3. A logic for ADR

We use a simple logic tailored on ADR. Basically, our logic is a propositional logic to predicate on (in)equalities of nodes. In the following we let $D, D', ...$ range over edges of $\Gamma$.

Definition 6 (ADR logic). Let $V$ be a countably infinite set of variables for nodes (ranged over by $x, y, z, ...$). The set $\mathcal{L}$ of (graph) formulae for ADR is given by the following grammar:

$$\psi, \varphi ::= x = y \mid \top \mid \neg \varphi \mid \varphi_1 \land \varphi_2 \mid \forall D(\bar{x}).\varphi$$

In formulae of the form $\forall D(\bar{x}).\varphi$, the occurrences of $y \in \bar{x}$ in $\varphi$ are bound, $\bar{x}$ has the length of the arity of $D$ and $\bar{x}$ are pairwise distinct.

Logic $\mathcal{L}$ is parametrised with respect to the type graph $\Gamma$ used in quantification. Variables not in the scope of a quantifier are free and the set $fv(\varphi)$ of free variables of $\varphi \in \mathcal{L}$ is defined accordingly; also, we abbreviate $x_1 = x_2 \land \ldots \land$
\[ x_{n-1} = x_n \text{ with } x_1 = x_2 = \ldots = x_{n-1} = x_n \] and we define \( \bot \) as \( \neg \top \), \( x \neq y \) as \( \neg(x = y) \), \( \varphi \lor \psi \) as \( \neg(\neg\varphi \land \neg\psi) \), \( \varphi \to \psi \) as \( \neg\varphi \lor \psi \), and \( \exists D(\bar{x})\varphi \) as \( \neg\forall D(\bar{x})\neg\varphi \).

The models of \( \mathcal{L} \) are ADR graphs together with an interpretation of the free variables of formulae.

**Definition 7 (Satisfaction relation).** An ADR graph \( G \) satisfies \( \varphi \in \mathcal{L} \) under the assignment \( h : V \to V \) (in symbols \( G \models h \varphi \)) iff

\[
\begin{align*}
\varphi &\equiv \top, & \text{or} \\
\varphi &\equiv x = y \quad \text{and} \quad h(x) = h(y), & \text{or} \\
\varphi &\equiv \neg\varphi' \quad \text{and} \quad G \models \neg h \varphi', & \text{or} \\
\varphi &\equiv \varphi_1 \land \varphi_2 \quad \text{and} \quad G \models_h \varphi_1 \text{ and } G \models_h \varphi_2, & \text{or} \\
\varphi &\equiv \forall D(\bar{x})\varphi \quad \text{and} \quad G \models_{h[\bar{x} \mapsto \bar{u}]} \varphi \quad \text{for any } d(\bar{u}) \in G \quad \text{s.t. } \tau_G(d) = D
\end{align*}
\]

Note that in the last clause of Definition 7, each bound variable in \( \bar{x} \) is replaced with a node.

**Fact.** For each \( h, h' : V \to V \), if \( h|_{\text{fv}(\varphi)} = h'|_{\text{fv}(\varphi)} \) then \( G \models_h \varphi \text{ iff } G \models_{h'} \varphi \).

By the above property, in \( G \models_h \varphi \) we can restrict to finite mappings \( h \) that only assign variables in \( \text{fv}(\varphi) \). Hereafter, we write \( G \models \varphi \) when \( \text{fv}(\varphi) = \emptyset \).

**Example 5.** The formula \( \varphi_{ex} = \forall D(x, y).\exists D'(z).x = z \) describes graphs such that each edge of type \( D \) is connected to one of type \( D' \) on the first tentacle. For instance, consider the graphs

\[
\begin{align*}
G_{valid} = \quad & \bullet \quad \xrightarrow{d_1 : D} \quad \bullet \quad \xrightarrow{d' : D'} \\
G_{invalid} = \quad & \bullet \quad \xrightarrow{d_1 : D} \quad \bullet \quad \xrightarrow{d' : D'} \\
& \bullet \quad \xrightarrow{d_2 : D} \quad \bullet \quad \xrightarrow{d' : D'}
\end{align*}
\]

then \( G_{valid} \) satisfies \( \varphi_{ex} \) whereas \( G_{invalid} \) does not because \( d_2 \) is not connected to any edge of type \( D' \).

More interesting formulae are given in the next two examples.

**Example 6.** The formula

\[
\text{noEdge}(D) \overset{\text{def}}{=} \forall D(\bar{x}).\bot
\]  

(3)

characterises the graphs that do not contain edges of a given type.

Formulae of the form (3) will be used in Definition 12 (hereafter, we write \( \text{noEdge}(D_1, \ldots, D_n) \) for \( \text{noEdge}(D_1) \land \ldots \land \text{noEdge}(D_n) \)).

The next example shows that, despite its simplicity, our logic is quite expressive when “taken modulo productions”.

**Example 7.** By the production below, a non-terminal edge of type \( C \) can be replaced by a chain of two edges of type \( C \). The formula \( \text{path } D \ C \) requires
instead that any two different nodes attached to an edge of type $D$ are connected by an edge of type $C$.

The production and the formula above characterise graphs that contain paths of edges of type $C$ between any two distinct nodes connected by an edge of type $D$. Note that even though there is no edge of type $D$ in the production, $\text{path } D C$ quantifies over edges of type $D$ in the graph.

4. Design by Contract for ADR

Our notion of contracts hinges on asserted productions, namely ADR productions decorated with pre- and post-conditions expressed in the logic $L$ given in §3. Hereafter, we fix an ADR production $p = (L, R, i)$.

**Definition 8 (Asserted productions).** Let $\psi, \varphi \in \mathcal{L}$ and $h, h' : V \to \mathcal{N}$ be two assignments. An expression of the form

$$\{\psi, h\} p \{\varphi, h'\}$$

where $h(\text{fv}(\psi)) \subseteq V_L$ and $h'(\text{fv}(\varphi)) \subseteq V_R$

is an asserted production.

An asserted production generalises ADR productions and it intuitively requires that if $p$ is applied to a graph $G$ that satisfies $\psi$ then the resulting graph is expected to satisfy $\varphi$. The maps $h$ and $h'$ in Definition 8 allow pre- and post-conditions to predicate on nodes occurring in the LHS or the RHS of $p$.

An instance $G'$ of a graph $G$ is a graph $G'$ isomorphic to $G$ that does not share nodes or edges with $G$. The application of an asserted production to a graph consists of replacing an homomorphic image of the edge of the LHS with a new instance of the RHS and then connecting it to the interface nodes. This is formalised in the next definition and schematically illustrated in Figure 1.

Figure 1: Asserted design productions
Definition 9 (Applying asserted productions). Let $G$ be a graph and $\sigma$ be a morphism from $L$ (the LHS of $p$) to $G$. An asserted production $\pi = \{\psi, h\} p \{\varphi, h'\}$ is applicable to $G$ via $\sigma$ iff $G \models_{\sigma h} \psi$.

Given an instance $R'$ of $R$ through the isomorphism $\iota : R \rightarrow R'$ such that $E_{R'} \cap E_G = \emptyset$ and $V_{R'} \cap V_G = \emptyset$ a graph $G[\sigma(e) \mapsto R']$ is the application of $\pi$ to $G$ with respect to $\sigma$ iff $R'' = R'[i(r) \mapsto \sigma(i^{-1}(r)) \mid r \in \text{Im}(i)]$. A production $\pi$ is valid when any application of $\pi$ to a graph satisfying the precondition of $\pi$ yields a graph satisfying the post condition of $\pi$.

Examples 8 and 9 show how asserted productions are applied to graphs.

Example 8. Consider the production bookFlight given in Example 3 and the formula and the asserted production

$$\psi \overset{\text{def}}{=} \forall \text{Fls}(x, y). x \neq y \quad \text{and} \quad \pi \overset{\text{def}}{=} \{\psi, \emptyset\} \text{bookFlight } \{\top, \emptyset\}$$

Then, $\pi$ cannot be applied to the leftmost graph $G$ in the rewriting of Example 4 because $G \not\models \psi$ (under the unique morphism $\sigma$ from $L$ to $G$). In fact, $x$ and $y$ are mapped to the same node $u_1$ of $G$.

Example 9. The rewriting below is obtained by applying $\pi$ in Example 8.

According to Definition 9, the edge fls on the left is replaced by an isomorphic instance of $R$ preserving the interface nodes $u_1$ and $u_3$.

Note that Definition 9 generalises the hyper-edge replacement mechanism of ADR; in fact, $\{\top, \emptyset\} p \{\top, \emptyset\}$ applies exactly as normal ADR productions.

5. Extracting contracts for ADR productions

The application of an asserted production $\{\psi, h\} p \{\varphi, h'\}$ to a graph satisfying $\psi$ does not necessarily yield a graph satisfying $\varphi$ (this can be trivially noted by taking a production with $\bot$ as post-condition). We give an algorithm to compute the weakest pre-condition given a post-condition and a production in the style of the seminal work on predicate transformers of Dijkstra [5]. Hereafter, bound variables in a formula are assumed distinct from its free variables and bound only once. We first give some auxiliary definitions and notations.

Let $\mathcal{N}$ denote the set of natural numbers. We use environments $\mathcal{E}$ to record how variables are quantified in a logical formula.

Definition 10 (Environments). An environment $\mathcal{E}$ is the product of three finite partial maps

$$\mathcal{E}^{(1)} : V \rightarrow \{\forall, \exists\} \times \mathcal{N}, \quad \mathcal{E}^{(2)} : V \rightarrow E_G \times \mathcal{N}, \quad \text{and} \quad \mathcal{E}^{(3)} : V \rightarrow \mathcal{N}$$

such that the following conditions hold:
• if $E^{(1)}$ is defined on $x$, so are $E^{(2)}$ and $E^{(3)}$
• if $E^{(2)}(x) = (D, h)$ then $1 \leq h \leq d$ where $d$ is the arity of $D$
• if $x_1 \neq x_2$, $E^{(1)}(x_j) = (q_j, l_j)$, and $E^{(2)}(x_j) = (D_j, h_j)$ for $j = 1, 2$ then $l_1 = l_2$ implies $q_1 = q_2$ and $h_1 \neq h_2$.

Let $E^{(i)}(x) \uparrow$ denote the fact that $E^{(i)}$ is undefined on $x$ (for $i = 1, 2, 3$) and $0$ denote the empty environment.

To avoid cumbersome parentheses, $\exists^l$ shortens $(q, l) \in \{\forall, \exists\} \times N$ (and similarly $\exists^h$ abbreviates $(D, h)$), moreover we ignore the indexes $l$ and $h$ when immaterial. An environment stipulates if $x \in V$ is quantified or not and, when it is, it yields the details of the quantification and the interpretation of $x$. More precisely,

• if $E^{(1)}(x) = \exists^l$ for some $(q, l) \in \{\forall, \exists\} \times N$, then $E^{(1)}$ specifies if $x$ is universally or existentially quantified, and that it is bound by the $l$-th quantifier,

• $E^{(2)} = \exists^h$ specifies the type $D$ and the tentacle $h$ of the edge $x$ is attached to in the quantification, and

• $E^{(3)}$ assigns a node $x$.

It is convenient to write $E(x) \equiv q \ D \ G$ when $x$ is quantified by $q$ (that is $E^{(1)}(x) = \exists^l$ for some $l \in N$), attached to an edge of type $D$ (that is $E^{(2)}(x) = D$), for some $h \in N$, and mapped to a node of $G$ (that is $E^{(3)}(x) \in V_G$); if $G$ consists of a node $n$, we simply write $E(x) \equiv q \ D \ n$. Also, we use "\" as a wild-card writing e.g. $E(x) \equiv q \ \_ \ G$ when the type assigned to $x$ is not defined or it is immaterial (i.e., $E(x) \equiv q \ \_ \ G$ abbreviates $E^{(1)}(x) = q$ and $E^{(3)}(x) \in V_G$).

In Definitions 11 and 12 below $p = (L, R, i)$ is a production and we write $R_e \equiv V_R \setminus \text{Im}(i)$ to denote the internal nodes of $p$, and $R^e \equiv R \setminus V_R$ to denote the nodes outside $p$. Also, $\text{cond}_L$ establishes the condition for the computation of the weakest pre-condition for $x_1 = x_2$ to return $\bot$; $\text{cond}_L$ is defined as the disjunction of the following cases:

1. $E(x_1) \equiv \exists \ \_ \ n, E^{(1)}(x_2) \uparrow$ and $n \notin R^e$
2. $E(x_1) \equiv \exists \ \_ \ R^e, E(x_2) \equiv \forall \ \_ \ n$ and $n \notin R^e$
3. $E(x_1) \equiv \exists \ \_ \ R^e, E(x_2) \equiv \exists \ \_ \ R^e$ and $E^{(3)}(x_1) \neq E^{(3)}(x_2)$
4. $E(x_1) \equiv \exists \ \_ \ R^e$, and either $E^{(1)}(x_2) \uparrow$ or $E(x_2) \equiv \forall \ \_ \ R^e$ and $E^{(3)}(x_1) \neq E^{(3)}(x_2)$
5. $E(x_1) \equiv \exists \ \_ \ n, E(x_2) \equiv \forall \ \_ \ R^e$ and $n \notin R^e$
6. $E(x_1) \equiv \forall \ \_ \ n, E^{(1)}(x_2) \uparrow$ and either $n \in R^e$ and $n \notin E^{(3)}(x_2)$ or $n \in \text{Im}(i)$
7. $E(x_1) \equiv \forall \ \_ \ R^e$ and either $E(x_2) \equiv \exists \ \_ \ R$ or $E(x_2) \equiv \exists \ \_ \ \text{Im}(i)$
8. $E(x_1) \equiv \forall \ \_ \ D \ \text{Im}(i), E(x_2) \equiv \exists \ \_ \ R$ and for an $e \in E_R$, $e$ has type $D$
9. $E(x_1) \equiv \forall \ \_ \ D \ \text{Im}(i), E(x_2) \equiv \exists \ D' \ \text{Im}(i)$ and there is $e \in E_R$ such that $e$ has type $D$ and any $e' \in \text{E}_R$ has a type different from $D'$
We comment each case above.

Definition 12 below will exploit the fact that free variables of the post conditions are mapped to nodes in the RHS \( R \) of \( p \) (cf. Definition 8). Therefore, case 1 above states that it is impossible to guarantee \( x_1 = x_2 \) when \( x_1 \) is existentially quantified and assigned to a node not in \( R \) and \( x_2 \) is a free variable of the post condition \( (\mathcal{E}(1)(x_2) \uparrow) \) because, by constructions such nodes would be different. For the same reason, in cases 2, 3, 4, and 5 above, \( \perp \) is returned when \( x_1 \) is assigned to an internal node of \( R \) while \( x_2 \) is either outside \( R^o \) or it is different from the node assigned to \( x_1 \). Case 6 stipulates that \( x_1 = x_2 \) cannot hold when either (as before) \( x_1 \) is assigned to an internal node of \( R \) while \( x_2 \) is assigned to a different node or because one node is assigned to (the image of) an node and the other to an internal. Cases 7 and 8 are similar: internal and external or (the image of) interface nodes of \( R \) cannot be the same. Case 9 states that if \( x_1 \) is universally quantified, has type \( D \), and assigned to a node in the interface of \( p \), then it cannot be identified with a variable \( x_2 \) assigned to an interface node of \( p \) attached to an edge whose type is not in \( R \).

**Definition 11 (Auxiliary map).** Given \( \psi_1, \psi_2 \in \mathcal{L} \) and \( \bar{\psi} \) equal to \( \mathcal{E}(3)(x_1) = \mathcal{E}(3)(x_2) \), \( eq_{p, x_1 = x_2}(\mathcal{E}) \) is defined as

\[
eq_{p, x_1 = x_2}(\mathcal{E}) = \begin{cases} \bar{\psi}, & \text{if } \mathcal{E}(1)(x_1) \uparrow, \mathcal{E}(1)(x_2) \uparrow \text{ or } \mathcal{E}(x_1) \equiv \exists \, R^o \text{ and } \mathcal{E}(1)(x_2) \uparrow \\ \perp, & \text{if } \text{cond}_\perp \\ \top, & \text{if } \mathcal{E}(x_1) \equiv \exists \, n, \mathcal{E}(x_2) \equiv \exists \, n \text{ and } n \in R^o \\ \top, & \text{if } \mathcal{E}(x_1) \equiv \forall \, n, \mathcal{E}(x_2) \equiv \exists \, n \text{ and } n \in R^o \\ \psi_1, & \text{if } \mathcal{E}(1)(x_1) \uparrow \text{ and } \mathcal{E}(x_2) \equiv \forall \, R^o \\ \psi_1, & \text{if } \mathcal{E}(x_1) \equiv \forall \, R^o \text{ and } \mathcal{E}(x_2) \equiv \forall \, R^o \\ \psi_1, & \text{if } \mathcal{E}(x_1) \equiv \exists \, n \text{ and } \mathcal{E}(x_2) \equiv \forall \, n \text{ and } n \in R^o \\ \psi_2, & \text{if } \mathcal{E}(x_1) \equiv \forall \, n \text{ and } \mathcal{E}(x_2) \equiv \forall \, n \text{ and } n \in R^o \\ x_1 = x_2, & \text{otherwise} \end{cases}
\]

and, depending on \( \mathcal{E} \), returns either \( \psi_1, \psi_2, x_1 = x_2, \top, \) or \( \perp \).

The map \( eq_{p, x_1 = x_2}(\mathcal{E}) \) in Definition 11 is parametrised with \( \psi_1 \) and \( \psi_2 \) and

- \( eq_{p, x_1 = x_2}(\mathcal{E}) \) returns \( \top \) when in \( \mathcal{E} \) both \( x_1 \) and \( x_2 \) are assigned to internal nodes of \( R \) (the RHS of \( p \)) since the application of \( p \) guarantees \( x_1 = x_2 \) for these cases;
- \( eq_{p, x_1 = x_2}(\mathcal{E}) \) returns \( \perp \) when one of the conditions 1, \ldots, 9 above holds;
- in the other cases, \( eq_{p, x_1 = x_2}(\mathcal{E}) \) returns either \( \psi_1, \psi_2, \) or \( x_1 = x_2 \); as more clear after Definition 12, such conditions state the absence of some edges from the graph \( p \) is applied to or the validity of a suitable equality.
A formula $\varphi \in \mathcal{L}$ is in negation normal form when negations occur only in front of equalities. Conventionally, in an equality $x = y$ occurring in $\varphi$, the quantification on $x$ (if any) precedes the one of $y$ (if any). It is trivial to see that all formulae of $\mathcal{L}$ have an equivalent negation normal form.

**Definition 12 (Weakest pre-conditions).** Let $\mathcal{Z} = \{z_1, \ldots, z_m\} \subseteq \mathcal{V}$ where $m$ is the arity of $L$, $\varphi \in \mathcal{L}$ in negation normal form, $h : \text{fv}(\varphi) \to \mathcal{V}_R$ be injective, and $\tilde{h} : \mathcal{Z} \to \mathcal{V}_L$ a bijection. Say that an environment $E$ is compatible with $h$ iff, for each $x \in X$, $\mathcal{E}^{(3)}(x) = \tilde{h}(x)$.

Given an environment $E$ compatible with $h$, the weakest pre-condition of $p$ with post-condition $\varphi$ under $h$, $h$, and $E$, denoted by $W_{h,E}^p(p, \varphi)$, is the formula $\text{wd}^{p,\top}_{\mathcal{L}}(\varphi) \land \text{wp}^{p,h}_{\mathcal{L},E}(\varphi)$ where in the definitions of $\text{wd}$ and $\text{wp}$ below we stipulate

- in the clauses for equality $x_1 = x_2$, $\mathcal{E}^{(2)}(x_1) = D_1$ and $\mathcal{E}^{(2)}(x_2) = D_2$;
- in the clauses for quantifiers $\forall D(x)_-$ and $\exists D(x)_-$ $D$ has arity $d$ and $\{v_1, \ldots, v_d\} \subseteq R$ is a fixed set of $d$ (representative) external nodes;
- condition $\tilde{u}$ on $R \cdot D$ holds iff $\tilde{u} \cap R^o = \emptyset$ when $R$ does not have edges of type $D$ and $\tilde{u} = u_1, \ldots, u_d$.

The function $\max \{l' \in \mathbb{N} \mid \exists y \in \mathcal{V} : \mathcal{E}^{(1)}(y) \vdash_{l'} \varphi \lor \mathcal{E}^{(1)}(y) \vdash_{l'} \exists \psi \} \text{ returns the maximum index of quantifiers in } E$ (we tacitly assume that $\max \emptyset = 0$).

Below $\text{eq}_{p,x_1 = x_2,\psi_3}(E)$ is as the auxiliary map $\text{eq}_{p,x_1 = x_2,\psi_3}(E)$ in Definition 11 but for the last case where instead of returning $x_1 = x_2$, $\text{eq}_{p,x_1 = x_2,\psi_3}(E)$ returns $\psi_3$. 

\[
\begin{align*}
\text{wd}^{p,\psi}_{\mathcal{L}}(x_1 = x_2) &= \text{eq}_{p,x_1 = x_2,\psi}(E) \\
\text{wd}^{p,\psi}_{\mathcal{L}}(x_1 \neq x_2) &= \begin{cases} 
\neg \text{eq}_{p,x_1 = x_2,\psi}(E) & \text{if } \mathcal{E}^{(1)}(x_1) \vdash_{1} \forall \exists R^o \text{ and } \mathcal{E}^{(3)}(x_1) = \mathcal{E}^{(3)}(x_2) \\
\neg \text{eq}_{p,x_1 = x_2,\psi}(E) & \text{otherwise}
\end{cases} \\
\text{wd}^{p,\psi}_{\mathcal{L}}(\top) &= \top \\
\text{wd}^{p,\psi}_{\mathcal{L}}(\bot) &= \begin{cases} 
\bot & \text{if } \{x \mid \mathcal{E}(x) \vdash_{1} \forall \exists R^o \} \neq \emptyset \\
\top & \text{otherwise}
\end{cases} \\
\text{wd}^{p,\psi}_{\mathcal{L}}(\phi \land \psi') &= \text{wd}^{p,\psi}_{\mathcal{L}}(\phi) \land \text{wd}^{p,\psi}_{\mathcal{L}}(\psi') \\
\text{wd}^{p,\psi}_{\mathcal{L}}(\phi \lor \psi') &= \text{wd}^{p,\psi}_{\mathcal{L}}(\phi) \lor \text{wd}^{p,\psi}_{\mathcal{L}}(\psi') \\
\text{wd}^{p,\psi}_{\mathcal{L}}(\forall D(x), \phi) &= \bigwedge_{\tilde{u} \text{ on } R \cdot D} \text{wd}^{p,\psi}_{\mathcal{L}}(\phi) \\
\text{wd}^{p,\psi}_{\mathcal{L}}(\exists D(x), \phi) &= \bigvee_{\tilde{u} \text{ on } R \cdot D} \text{wd}^{p,\psi}_{\mathcal{L}}(\phi)
\end{align*}
\]
The weakest pre-condition is the conjunction of the predicates computed by the predicate transformers \(\wp_{E,h}^p\) and \(\wp_{E,h}^p\) on the post-condition \(\varphi\). The transformer \(wd\) checks if the RHS of a production satisfies the post-condition. This will be useful to accommodate the cases where the post-condition requires the existence of edges with some properties that the RHS of production ensures; then the post-condition is guaranteed whatever graph the production is applied to. The most interesting cases in Definition 12 are the ones for equality \(x_1 = x_2\) dealt by the auxiliary map \(eq_{p,x_1=x_2}^p\).

- If both \(x_1\) and \(x_2\) are existentially quantified and assigned to the same internal nodes of \(p\), the calculated weakest pre-condition is \(\top\); in fact, whatever graph the production is applied to, the post-condition would be guaranteed by the RHS of \(p\).

- Instead \(\bot\) is returned when say \(x_1\) is universally quantified and

  (i) either \(x_2\) is existentially quantified and assigned to an interface node

  (ii) or it is assigned to an internal node of \(R\) different from the one assigned to \(x_2\).

Note that in (i) if \(x_2\) were universally quantified, there might be a chance to guarantee the equality if no edges of the type quantifying the variables were in the graph \(p\) is applied to. In fact, \(eq_{p,x_1=x_2}^p\) returns \(\bot\) if (i) \(x_1\) is mapped to a fresh node in the RHS of \(p\) (i.e., an internal node of \(p\)) while \(x_2\) is mapped to a node outside \(p\) or (ii) if they are mapped to two fresh nodes of the RHS of \(p\) because the semantics of ADR does not allow such identifications on the internal nodes of a production.
The equality $x_1 = x_2$ may hold if $x_1$ and $x_2$ are mapped on the same internal node provided that no edge in the graph $p$ is applied to is typed as the type of the edges insisting on the variables, otherwise the universal quantification will be spoiled.

Likewise, if both variables are universally quantified but one is internal and the other is external (not in $p$), then the weakest pre-condition returns $\text{noEdge}(D)$ where $D$ is the type of the external variable. Intuitively, the graph resulting from the application of $p$ to a graph with an $e$ edge of type $D$, would violate the quantification of $x_1$ and $x_2$ since $e$ cannot insist on fresh nodes introduced by $p$.

In all other cases, $wp_{E,h}^{p,\bar{E}}(x_1 = x_2)$ requires the initial graph to satisfy the same equality on the nodes corresponding to the variables of the post-condition; this requires that if either $x_1$ and $x_2$ are assigned to an interface node (that is $h(x_j) \in Im(i)$) it has a counterpart variable $z \in \{z_1, \ldots, z_m\}$ mapped (through $\bar{h}$) on the node $i^{-1}(x_1)$ or $i^{-1}(x_2)$ in $L$.

The remaining cases are trivial but for the quantifications $\forall D(x), \phi$ and $\exists D(x), \phi$ where the computed pre-conditions require $\phi$ to be satisfied under any “reasonable” assignment to $\bar{x}$ for the universal quantification or one “reasonable” assignment to $\bar{x}$ for the existential quantification; this means that such variables are assigned in any possible way either to nodes in $R$ or to a fixed set of nodes $v_1, \ldots, v_d$ outside $R$. The identity of $v_1, \ldots, v_d$ is immaterial, in fact, the crucial point is that they refer to nodes outside $R$ (i.e., as many as the variables in $\bar{x}$).

**Proposition 1.** If $\psi$ and $\psi'$ are logically equivalent $\mathcal{L}$-formulae, then $wd_{E}^{p,\bar{E}}(\phi)$ (resp. $wp_{E,h}^{p,\bar{E}}(\phi)$) is logically equivalent to $wd_{E}^{p,\psi}(\phi')$ (resp. $wp_{E,h}^{p,\psi}(\phi')$).

The next example shows how to compute weakest pre-conditions.

**Example 10.** Consider $\phi \in \mathcal{L}$ and the production $p$ below:

$$\phi \overset{\text{def}}{=} \forall B(x,y).\forall C(z).y = z \quad p \overset{\text{def}}{=} \begin{array}{c}
\text{A} \\
\text{u} \\
\text{b : B} \quad \text{w} \\
\end{array}$$

Let $\bar{h} = w \mapsto v$. The first step (i) computes $wd_{E}^{\bar{E},0}(p, \phi) \overset{\text{def}}{=} wd_{0}^{p,\top}(\phi) \land wp_{E,h}^{p,\phi}(\phi)$ and (ii) applies the quantification case in Definition 12 so to yield

$$( \bigwedge_{j=1,2,3} wd_{E,j}^{p,\psi}(\phi') ) \land ( \bigwedge_{j=1,2,3} \forall B(x,y). wp_{E,h}^{p,\phi}(\phi') )$$

given that $\mathcal{E}_1 = \{ x \mapsto (\forall, B, u_1), y \mapsto (\forall, B, u) \}$, $\mathcal{E}_2 = \{ x \mapsto (\forall, B, u_1), y \mapsto (\forall, B, v_1) \}$ and $\mathcal{E}_3 = \{ x \mapsto (\forall, B, v_1), y \mapsto (\forall, B, v_2) \}$ are the only assignments to consider (since $v_1$ and $v_2$ are representative nodes outside the RHS of $p$ while $u_1$ the unique node on the interface, and $u$ the unique internal node).
The second step applies again this case for \( \forall C(z) \) (for both \( \text{wp}^{p,h}_{E_j}(\varphi') \) and \( \text{wp}^{p,h}_{E_j,\emptyset}(\varphi') \)) and yields
\[
\left( \bigwedge_{j,k=4,5} \text{wp}^{p,h}_{E_j}(\varphi') \right) \land \left( \bigwedge_{j,k=4,5} \forall B(x,y), \forall C(z), \text{wp}^{p,h}_{E_j\cup E_k,\emptyset}(\varphi'') \right)
\]
where \( E_4 = \{ z \mapsto (\forall, C, u_1) \} \) and \( E_5 = \{ z \mapsto (\forall, C, v_1) \} \): in fact there is no edge of type \( C \) in the RHS of \( p \) (hence \( v_1 \) is representative external node and \( u_1 \) is its unique interface node).

Finally, applying the auxiliary map \( \text{eq}^{\psi_1,\psi_2}_{p,x_1,x_2}(E) \) for node equality, we get
\[
\bigwedge_{j,k} \text{wp}^{p,h}_{E_j\cup E_k}(\varphi') = (T \land \text{noEdge}(C)) \land (T \land T) = \text{noEdge}(C) \quad (4)
\]
\[
\bigwedge_{j,k} Q[\text{wp}^{p,h}_{E_j\cup E_k,\emptyset}(\varphi'')] = Q[\text{noEdge}(C)] \land Q[y = z] \quad (5)
\]
where in (5) \( Q[\cdot] \) is the context \( \forall B(x,y), \forall C(z)[\cdot] \). Note that, the weakest pre-conditions is the conjunction of (4) and (5), that is
\[
W^{h}_{\emptyset,E}(p,\phi) = \text{noEdge}(C) \land Q[\text{noEdge}(C)] \land Q[y = z]
\]
this is consistent with the fact that \( \phi \) can only be satisfied by graphs that do not have any edges of type \( C \) due to the internal node \( u \) introduced by \( p \).

The correctness of the algorithm is established by showing the validity of the asserted production \( \{ W^{h}_{h,E}(p,\varphi), h \} \) \( p \) \( \{ \varphi, h \} \) (Theorem 1) and that the \( W^{h}_{h,E}(p,\varphi) \) is the weakest pre-condition for \( p \) and \( \varphi \) (Theorem 2).

Function \( C \) takes \( \varphi \in \mathcal{L} \) and an environment \( E \) and returns a new formula that takes into account the quantifications in \( E \) of the free variables of \( \varphi \).

**Definition 13 (Formulae in context).** Let \( \varphi \in \mathcal{L} \) and \( E \) an environment as in Definition 10. Define
\[
C(E, \varphi) = C(E \setminus \{ y \mapsto \bar{h}_j' \mid y \in E, qD(x_1',\ldots,x_d'), \phi \}) \quad (6)
\]
where \( h = \max \{ h' \mid \exists y \in V, q' \in \{ \forall, \exists \} : E(y) = (q',\ldots,\bot) \} \)

and \( x'_j = \begin{cases} z & \text{if } z \in E \land E(z) \equiv h_j D \\ * & \text{otherwise} \end{cases} \)

and define \( C(E, \varphi) = \varphi, \text{ if } E = \emptyset \).

Notice that \( C \) is well defined as (6) in Definition 13 does not depend on the choice of the variables.

**Theorem 1.** Let \( p = (L, R, i) \) be a production, \( \varphi \in \mathcal{L} \), \( h : \text{fv}(\varphi) \to V_R \) be injective, \( \bar{h} : Z \to V_L \) be a bijection where \( Z \subseteq V \) disjoint from the variables in \( E \), \( E \) be an environment compatible with \( h \), and \( \pi \) be the asserted production \( \{ W^{h}_{h,E}(p,\varphi), h \} \) \( p \) \( \{ \varphi, h \} \). For any ADR graph \( G \) and the morphism \( \sigma \) from \( L \) to \( G \), if \( G \setminus \sigma(E_L) \models_E W^{h}_{h,E}(p,\varphi) \) then \( \pi(G, \sigma) \models_E C(E, \varphi) \).
Proof (sketch) The proof is by induction on the structure of $\varphi$. The details are relegated in Appendix A.

Theorem 2. Given $\varphi \in \mathcal{L}$, let $Z \subseteq V$ such that no variables in $\varphi$ is in $Z$ and $h : Z \rightarrow V_L$ be a bijection, then for any formula $\psi$ such that $\{\psi, h\} \vdash \{\varphi, h\}$ is a valid production for any graph $G$ then $\psi$ implies $\mathcal{W}^h_{\varphi, \epsilon}(p, \varphi)$.

Proof (sketch) The proof is by induction on the structure of $\mathcal{W}^h_{\varphi, \epsilon}(p, \varphi)$. The details are relegated in Appendix B.

6. Applying Design Contracts

We show how asserted productions can guarantee design properties by considering a simplified version of Java’s remote method invocation (JRMI) mechanism and show how semantic conditions of JRMI can be modelled by asserted ADR productions. The asserted productions are obtained by using the algorithm in § 5.

Figure 2 shows the type graph for JRMI which includes types as follows:

<table>
<thead>
<tr>
<th>Edges</th>
<th>Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\bigcirc$ non-remote objects</td>
<td>$\bigcirc$ inheritance relation</td>
</tr>
<tr>
<td>$\text{RO}$ remote objects</td>
<td>$\bullet$ method overriding</td>
</tr>
<tr>
<td>$\text{RI}$ remote interfaces</td>
<td>$\otimes$ implemented remote interfaces</td>
</tr>
<tr>
<td>$\text{URO}$ class UnicastRemoteObject</td>
<td>$\bigcirc$</td>
</tr>
<tr>
<td>$\text{OM}$ equals and hashCode methods</td>
<td>$\bigcirc$</td>
</tr>
</tbody>
</table>

The intuition is that Java objects are either in non-remote classes $\text{O}$ or in $\text{RO}$ ones. In the latter case, we want them to provide specific methods (i.e., equals and hashCode) to allow objects to be exported. Recall that in JRMI a remote object has to be exported in order to make its methods remotely invokable. This can be done in two ways: (i) by extending the class UnicastRemoteObject or (ii) by explicitly implementing some of the java.lang.Object methods. To account for this, we use $\text{OM}$ and the node type $\bullet$ that explicitly models the overriding.
The productions of our model are in Figure 3. The capability of Java to permit to make local invocations to remote methods when those are not exported, is captured by the production obj2RUUsingURO. Moreover, we provide productions to capture (i) and (ii) above. These possibilities are covered by the first tree productions obj2RUUsingURO and extUnicast in Figure 3 while the last two productions extUnicast and addMethod are for remote objects design.

The following invariant establishes that every remote object either extends URO or implements an OM

\[ \phi_{inv} \overset{def}{=} \forall \text{RO}(x, y, z, w). (\exists \text{URO}(a, b). x = b \lor \exists \text{OM}(c). w = c) \]

which intuitively imposes designers to avoid reusing remote objects for non-remote ones.

We now apply our algorithm which requires to compute the weakest precondition for every production in the system. We initially consider productions extUnicast and obj2RUUsingURO. By Definition 12, the algorithm has to compute

\[ \forall_{\theta, 0}(p, \phi_{inv}) = wd^{p, \top}_{\theta, 0}(\phi_{inv}) \land wp^{p, \checkmark}_{\theta, 0}(\phi_{inv}) \]

Therefore, the case of universal quantification is used first:

\[ (\bigwedge_{j=1, \ldots, 4} wd_{E_j}^{p, \top}(\phi_{inv}) ) \land (\bigwedge_{j=1, \ldots, 4} \forall \text{RO}(x, y, z, w). wp_{E_j, \theta}(\phi'_{inv}) ) \]
where \( \phi_{inv}^i = \exists \text{URO}(a, b).x = b \lor \exists \text{OM}(c).w = c \). Let \( v_1 \) and \( v_2 \) be representative nodes outside the RHS of the productions above and let \( u_1, u_2, \) and \( u \) be as in such production in Figure 3 (that is \( u_1 \) and \( u_2 \) are interface nodes and \( u \) the internal node). The assignments

\[
E_1 = \{ x \mapsto (\forall, \text{RO}, u), \ y \mapsto (\forall, \text{RO}, u_1), \ z \mapsto (\forall, \text{RO}, u_2), \ w \mapsto (\forall, \text{RO}, u_3) \}
\]

\[
E_2 = \{ x \mapsto (\forall, \text{RO}, u_1), \ y \mapsto (\forall, \text{RO}, u_2), \ z \mapsto (\forall, \text{RO}, u_3), \ w \mapsto (\forall, \text{RO}, u_4) \}
\]

\[
E_3 = \{ x \mapsto (\forall, \text{RO}, v_1), \ y \mapsto (\forall, \text{RO}, v_2), \ z \mapsto (\forall, \text{RO}, v_3), \ w \mapsto (\forall, \text{RO}, v_4) \}
\]

\[
E_4 = \{ x \mapsto (\forall, \text{RO}, u_1), \ y \mapsto (\forall, \text{RO}, v_1), \ z \mapsto (\forall, \text{RO}, v_2), \ w \mapsto (\forall, \text{RO}, v_3) \}
\]

cover all the cases (all the non listed cases are captured by \( E_3 \)) for the computation of \( \mathcal{W}_{h,0}^E(p, \phi_{inv}) \).

The next step requires the use of the disjunction rule returning

\[
wd_{E_j,x}^{p,T} (\phi''_{inv}) = \bigvee_{E_j,x} wd_{E_j,x}^{p,T} (\phi''_{inv})
\]

where \( \phi''_{inv} = \exists \text{URO}(a, b).x = b \) and \( \phi'''_{inv} = \exists \text{OM}(c).w = c \).

By applying now the existential quantification steps for \( \phi''_{inv} \) and \( \phi'''_{inv} \) we get

\[
wd_{E_j,x}^{p,T} (\phi_{inv}) = ( \bigvee_{E_j,x} wd_{E_j,x}^{p,T} (x = b) )
\]

\[
wd_{E_j,x}^{p,T} (\phi_{inv}) = ( \bigvee_{E_j,x} wd_{E_j,x}^{p,T} (w = c) )
\]

\[
p_{E_j,x}^{p,\phi,\emptyset} (\phi''_{inv}) = ( \bigvee_{E_j,x} \exists \text{URO}(a, b).wp_{E_j,x}^{p,\phi,\emptyset} (x = b) \lor \bigvee_{E_j,x} wd_{E_j,x}^{p,T} (x = b) )
\]

\[
p_{E_j,x}^{p,\phi,\emptyset} (\phi''_{inv}) = ( \bigvee_{E_j,x} \exists \text{OM}(c).wp_{E_j,x}^{p,\phi,\emptyset} (w = c) \lor \bigvee_{E_j,x} wd_{E_j,x}^{p,T} (w = c) )
\]

where \( k = 5, 6, 7 \) and \( l = 8, 9 \) and the assignments for \( E_5, \ldots, E_9 \) are:

\[
E_5 = \{ a \mapsto (\exists, \text{URO}, u_1), \ b \mapsto (\exists, \text{URO}, u) \}
\]

\[
E_6 = \{ a \mapsto (\exists, \text{URO}, u_1), \ b \mapsto (\exists, \text{URO}, u_2) \}
\]

\[
E_7 = \{ a \mapsto (\exists, \text{URO}, u_1), \ b \mapsto (\exists, \text{URO}, v_2) \}
\]

\[
E_8 = \{ c \mapsto (\exists, \text{OM}, u_1) \}
\]

\[
E_9 = \{ c \mapsto (\exists, \text{OM}, v_1) \}
\]

Finally, applying the node equality cases in the auxiliary map \( e_p^{\phi_1,\phi_2,\phi_3}(E) \) of Definition 12, we get

\[
\bigwedge_j \bigvee_{E_j,x} wd_{E_j,x}^{p,T} (\phi''_{inv}) = (T \lor \bot \lor \bot \land (T \lor T \lor \bot) \land \ldots = T \tag{7}
\]

\[
\bigwedge_j \bigvee_{E_j,x} wd_{E_j,x}^{p,T} (\phi''_{inv}) = (T \lor \bot \lor T \land (T \lor \bot \lor T) \land \ldots = T \tag{8}
\]
where, letting $Q[\cdot]$ (resp. $Q'[\cdot]$) be the context $\exists URO(a, b)[\cdot]$ (resp. $\exists OM(c)[\cdot]$),

\[
\begin{align*}
\phi &= (Q[x = b] \lor Q[\bot] \lor Q[\bot]) \land (Q[x = b] \lor \ldots) \land \ldots = Q[x = b] \\
\phi' &= (Q'[w = c] \lor Q'[\bot] \lor Q'[w = c]) \land (Q'[w = c] \lor \ldots) \land \ldots = Q'[w = c]
\end{align*}
\]

Using the computations above

\[
\phi_{\text{inv}} = W_{0,0}^h(p, \phi_{\text{inv}}) = \top \land \forall RO(x, y, z, w). (Q[x = b] \lor Q'[w = c])
\]

The weakest pre-condition for obj2RUsingOM and addMethod is $\phi_{\text{inv}}$ itself. Indeed, the productions do not violate the invariant. For a production to violate $\phi_{\text{inv}}$ it has to create a fresh edge of type RO that is not attached to any edge of either type URO or OM. By applying though the algorithm to production obj2R we get $W_{0,0}^h(obj2R, \phi_{\text{inv}}) = \bot$ as the production introduces a RO with no URO or OM attached to it. Cases 4 and 5 of the auxiliary mapping in Definition 11 forces the algorithm to return $\bot$. We omit the workings as $W_{0,0}^h(p, \phi_{\text{inv}})$ is computed in a similar fashion as in Example 10 and above for productions extUnicast and obj2RUsingURO.

By checking the weakest pre-condition against a graph before applying a production we guarantee that after the application the obtained graph will satisfy the architectural style specified by the invariant. This way we can use a more general design of a system like in this case and using the invariant stop the system from evolving independently without removing or adding new productions.

7. A methodology for recovering invalid configurations

In this paper, we envisage architectural styles as formalised by a set of ADR productions combined with a formula of our logic specifying an invariant of the system as illustrated in Example 11 below.

Example 11. Consider the run-time reconfiguration

\[
\begin{array}{c}
\begin{array}{c}
S \rightarrow \bullet \quad - \quad C \\
\end{array}
\end{array}
\quad \text{badServer()} 
\begin{array}{c}
\begin{array}{c}
\text{badServer()} \rightarrow \bullet \quad - \quad C \\
\end{array}
\end{array}
\]

where $S$ changes as illustrated to model a failure $F$. By imposing an invariant that states that every client has to be connected to a non-failed server, the invalid configuration can be identified and recovered.

We give a basic methodology for recovering a system to a valid state when run-time configurations compromise it. We will assume that ADR graphs may be subject to run-time changes. Instead of giving a formal definition for such
graph rewritings, for the sake of this paper it is enough to consider simple local rewritings whereby edges may become corrupted and in turn compromise the desired architectural style in terms of the specified invariant. In §10 we briefly discuss more complex methodologies that we plan to consider in the future developments.

We are interested in computations that start from a system configuration, say \( s_0 \), that corresponds to an initial graph, say \( G_0 \), supposed to satisfy the invariant, say \( \phi_{\text{inv}} \). The system may evolve at run-time through a series of reconfigurations \( (r_i) \) that are reflected at the architectural level as schematically represented in the diagram (10) below (where \( G_i \vdash s_i \) stands for '\( s_i \) can be parsed as \( G_i \)'):

\[
\begin{align*}
G_0 & \rightarrow G_1 \rightarrow \cdots \rightarrow G_{k-1} \rightarrow G_k \rightarrow \cdots \\
\top & \sim r_1 \sim s_1 \sim r_2 \sim \cdots \sim r_{k-1} \sim s_{k-1} \sim r_k \sim s_k \sim r_{k+1} \sim \cdots 
\end{align*}
\]

We assume that most of the run-time reconfigurations produce graphs that do not violate \( \phi_{\text{inv}} \). Occasionally, the graph obtained by a run-time reconfiguration, say \( G_i \), may violate \( \phi_{\text{inv}} \). Our approach essentially computes how to rewrite graph \( G_i \) to a graph \( G_{i+1} \) satisfying \( \phi_{\text{inv}} \) and then reflect this into \( s_i \) by means of reconfigurations leading to a state \( s_{i+1} \) with architecture \( G_{i+1} \).

We propose a simple methodology that can select a number productions that when applied to \( G_i \) induces a reconfiguration of the violating system into a state whose style satisfies \( \phi_{\text{inv}} \). We assume a monitoring mechanism that triggers our methodology whenever a reconfiguration yields to an invalid system.

Our methodology consists of the steps 1 ÷ 5 below that require the designer to specify the productions and an architectural invariant \( \phi_{\text{inv}} \) so to establish the architectural style of interest (as done in Example 11).

1. The architecture (say \( G \)) corresponding to the configuration of the current system is computed through ADR parsing.
2. Check that \( G \) satisfies \( \phi_{\text{inv}} \).
3. If \( G \not\models \phi_{\text{inv}} \) then, for each production \( p \), compute the weakest pre-condition \( \psi \) with respect to \( \phi_{\text{inv}} \).
4. Select a production \( p \) such that \( G \setminus \sigma(E_L) \models \psi \) (if any); apply \( p \) to \( G \) to determine the reconfiguration needed for the system to reach a valid state.
5. If the designer considers not satisfactory the reconfigured system obtained in the previous stage or if there is no production \( p \) such that \( G \setminus \sigma(E_L) \models \psi \), then the designer may repeat steps 3 and 4 by replacing \( G \) with \( G \setminus \sigma(E_L) \) and \( \phi_{\text{inv}} \) with \( \psi \).

In step 1, we rely on the parsing mechanism of ADR (cf. [3]) whereby productions can be used “backward” to retrieve the architecture of a configuration. For space limit, we do not present the parsing mechanism and refer the interested reader to [3]. In step 2, we assume that an underlying monitoring mechanism uses the \( \models \) relation of our logic to determine if the graph \( G \) computed in step 1
violates the invariant. In such case, step 3 uses the algorithm on each production to compute their weakest pre-conditions (this step does not need to be re-iterated at each reconfiguration). In step 4, if the architecture of the violating system satisfies one of the computed preconditions, such production is a candidate to establish a new architecture and trigger the appropriate reconfigurations on the invalid system. Finally, in step 5 the designer has to decide whether to stop or continue the process. In the latter case, the idea is to repeat steps 3 and 4 replacing $\mathcal{G}$ with $\mathcal{G} \setminus \sigma(E_L)$ and $\phi_{\text{inv}}$ with $\psi$ so to compute the weakest pre-condition of the weakest pre-condition computed in the previous iteration. This allows us to exploit every possible order the productions can be in order to fix the architectural style. Note that the morphism that invalidate $\mathcal{G} \models \phi_{\text{inv}}$ indicates which part of the system has to be rewritten, while the production $p$ suggests plausible reconfigurations.

In § 8 we apply the methodology above to a simple example.

8. Applying the methodology

We consider a scenario where a flight search engine allows users to book flights. We use the type graph in Example 2 where there is only one type of node $\bullet$ and the types of edges are $\mathcal{C}$ (for clients), $\mathcal{BF}$ (for the booking flights services), $\mathcal{FF}$ (for the broker service finding flights), $\mathcal{Fls}$ (for the different flights available), $\mathcal{Fl}$ (for the flight to be booked), and $\mathcal{P}$ and $\mathcal{PF}$ (for completed or failed payment services, respectively). Consider the following productions:

- **findFlights** establishes a broker service $\mathcal{FF}$,
- **bookFlight** yields a flight ($\mathcal{Fl}$) connected to a payment service ($\mathcal{P}$),
- **browseFlights** generates as many flights as necessary,
- **deleteFlight** and **noFlights** respectively remove and stop adding flights to the design.

Services can either be composed with other services using **findFlights** and **bookFlight** like for instance when one chooses a specific flight and the system needs to “invoke” another service (payment service) to complete the request, or branch using the production **browseFlights** to represent the different flights a customer can choose from.
Figure 4 shows the architectural style of a system where a client books a flight and successfully pays for it. Initially, the client searches for a flight by invoking the findFlight service which, in turn, invokes different airlines about their flights. Once a flight is selected a payment service is used to complete the transaction.

Sometimes, failures are possible during the payment; this is modelled in Figure 4(b) where the payment edge P reconfigures as a PF edge. We show how to apply our methodology in this scenario.

8.1. Fixing the Architectural Style Using a Single Production

The style we consider consists of the productions above and the invariant

\[ \phi_{F1} = \exists F1(x_1, x'_1). \exists P(x_2', x_2). x_1 = x_2 \]

specifying that some flight F1 has to be connected to a successful payment P.

Following the methodology presented in § 7, we need to check if graph \( G_b \) given in Figure 4(b) satisfies the invariant \( \phi_{F1} \) and find that \( G_b \not\equiv \phi_{F1} \). In fact, there is no edge of type P in \( G_b \) so we invoke \( W_{h,0}^P(p, \phi_{F1}) \) on every production \( p \) where \( h = \emptyset \) (since \( \phi_{F1} \) is a closed formula) and \( h \) maps the interface nodes of \( p \). We have \( W_{h,0}^P(p, \phi_{F1}) = \phi_{F1} \) for all \( p \neq \text{bookFlight} \) whereas, for \( p = \text{bookFlight} \), \( W_{h,0}^P(p, \phi_{F1}) = \top \).

We show that \( W_{h,0}^P(p, \phi_{F1}) \) acts in the same way (and yields \( \phi_{F1}^p \)) for any \( p \neq \text{bookFlight} \) since such productions do not have edges of type F1 or P in their RHS. We have to compute \( w_{\mathcal{E}_j}^p(\phi_{F1}) \) by first applying the case of existential quantification (cf. Definition 12):

\[
( \bigvee_{j=1}^{5} w_{\mathcal{E}_j}^p(\phi_{F1}) ) \land ( \bigvee_{j=1}^{5} \exists F1(x_1, x'_1). w_{\mathcal{E}_j,h}^{P,h}(\phi_{F1}) \lor w_{\mathcal{E}_j}^p(\phi_{F1}) )
\]

where \( \phi_{F2} = \exists P(x_2', x_2). x_1 = x_2 \). Let \( v_1 \) and \( v_2 \) be representative nodes outside the RHS of the productions above, \( u_1 \) and \( u_2 \) be interface nodes of the productions. The assignments

\[
\begin{align*}
\mathcal{E}_1 & = \{ x_1 \mapsto (\exists F1, u_1), \ x'_1 \mapsto (\exists F1, v_1) \} \\
\mathcal{E}_2 & = \{ x_1 \mapsto (\exists F1, u_2), \ x'_1 \mapsto (\exists F1, v_1) \} \\
\mathcal{E}_3 & = \{ x_1 \mapsto (\exists F1, v_1), \ x'_1 \mapsto (\exists F1, u_1) \} \\
\mathcal{E}_4 & = \{ x_1 \mapsto (\exists F1, v_1), \ x'_1 \mapsto (\exists F1, u_2) \} \\
\mathcal{E}_5 & = \{ x_1 \mapsto (\exists F1, v_1), \ x'_1 \mapsto (\exists F1, v_2) \}
\end{align*}
\]
are the only ones to consider for the first quantification. Instead, for the other existential quantification \( \exists P(x_2', x_2) \) yields

\[
\bigvee_{j,k=7, \ldots ,11} wd^p_{E_j \cup E_k}(\phi''_{F_2}) \quad \wedge \quad \bigvee_{j,k=7, \ldots ,11} Q[wp^p_{E_j \cup E_k}(\phi''_{F_2}), \emptyset] \vee wd^p_{E_j \cup E_k}(\phi''_{F_2})
\]

where \( \phi''_{F_2} \) is the equality \( x_1 = x_2 \), \( Q[\cdot] \) is the context \( \exists F_1(x_1, x_1') \). \( \exists P(x_2', x_2), [\cdot] \) and the assignments \( \mathcal{E}_7, \ldots ,\mathcal{E}_{11} \) are:

\[
\begin{align*}
\mathcal{E}_7 &= \{ x_2 \mapsto (\exists, P, u_1), \ x_2' \mapsto (\exists, P, v_1) \} \\
\mathcal{E}_8 &= \{ x_2 \mapsto (\exists, P, u_2), \ x_2' \mapsto (\exists, P, v_1) \} \\
\mathcal{E}_9 &= \{ x_2 \mapsto (\exists, P, v_1), \ x_2' \mapsto (\exists, P, u_1) \} \\
\mathcal{E}_{10} &= \{ x_2 \mapsto (\exists, P, v_1), \ x_2' \mapsto (\exists, P, u_2) \} \\
\mathcal{E}_{11} &= \{ x_2 \mapsto (\exists, P, v_1), \ x_2' \mapsto (\exists, P, v_2) \}
\end{align*}
\]

Finally, applying the case for node equality in the auxiliary map \( eq^{\psi_1,\psi_2,\psi_3}(\mathcal{E}) \) of Definition 12, we get

\[
\begin{align*}
\bigvee_{j,k} wd^p_{E_j \cup E_k}(\phi''_{F_1}) &= \top \vee \top \vee \cdots = \top \quad (11) \\
\bigvee_{j,k} Q[wp^p_{E_j \cup E_k}(\phi''_{F_1}), \emptyset] &= (\phi_{F_1} \vee \bot) \vee (\phi_{F_1} \vee \bot) \vee \cdots = \phi_{F_1}(12)
\end{align*}
\]

which yield \( W^h_{0,0}(p, \phi_{F_1}) \) since (11) and (12) respectively correspond to \( wd^p_{E}(\phi_{F_1}) \) and \( wp^p_{E}(\phi_{F_1}) \).

We now consider \( p = bookFlight \) and show that \( W^h_{0,0}(p, \phi_{F_1}) = \top \). As in the previous case, we consider the quantifications for which we have to consider the extra mappings due to \( F_1 \) and \( P \):

\[
\begin{align*}
\mathcal{E}_6 &= \{ x_1 \mapsto (\exists, F_1, u), x_1' \mapsto (\exists, F_1, u_2) \} \\
\mathcal{E}_{12} &= \{ x_2 \mapsto (\exists, P, u_1), x_2 \mapsto (\exists, P, u) \}
\end{align*}
\]

where \( u_1 \) and \( u_2 \) are the production’s interface nodes as before and \( u \) is its unique internal node. By the quantification cases we have

\[
\begin{align*}
\bigvee_{j,k} wd^p_{E_j \cup E_k}(\phi''_{F_1}) \quad \wedge \quad \bigvee_{j,k} Q[wp^p_{E_j \cup E_k}(\phi''_{F_2}), \emptyset] \vee wd^p_{E_j \cup E_k}(\phi''_{F_1})
\end{align*}
\]

where \( j = 1, \ldots ,6 \) and \( k = 7, \ldots ,12 \).

Finally, applying the case for node equality in the auxiliary map \( eq^{\psi_1,\psi_2,\psi_3}(\mathcal{E}) \) of Definition 12, we get

\[
\begin{align*}
\bigvee_{j,k} wd^p_{E_j \cup E_k}(\phi''_{F_1}) &= \top \vee \top \vee \cdots = \top \quad (13) \\
\bigvee_{j,k} Q[wp^p_{E_j \cup E_k}(\phi''_{F_2}), \emptyset] &= (Q[\top] \vee \top) \vee \cdots = \top \quad (14)
\end{align*}
\]

23
Note that the weakest pre-conditions is the conjunction of (13) and (14), that is
\( \text{wp}_0^b(b) \land \text{wp}_0^L(b) = \top \)

The next step requires that we check whether the graph \( G_b \) given in Figure 4(b) satisfies any of the weakest pre-conditions computed.

\( G_b \not\vdash \exists F1(x_1, x_1, \exists P(x_2'). x_1 \neq x_2 \) but instead \( G_b \models \top \) and therefore we know that by applying the production \text{bookFlight} we get a graph \( G_b' \) that satisfies the invariant \( \phi_{\text{F1}} \).

8.2. Fixing the Architectural Style Using Multiple Productions

Let us now consider a more strict invariant where:

\[ \phi_{\text{F1}} = \forall F1(x_1, x_1'). \exists P(x_2'). x_1 = x_2 \land \forall PF(x_3, x_3'). \bot \]

that specifies in this case that every flight \( F1 \) has to be connected to a successful payment \( P \) and also there shouldn’t exist any edge of type \( PF \) in the graph.

Following the methodology presented in § 7, we need to check if graph \( G_b \) given in Figure 4(b) satisfies the invariant \( \phi_{\text{F1}} \) to find that \( G_b \not\models \phi_{\text{F1}} \). In fact, there is no edge of type \( P \) in \( G_b \) and also there is an edge of type \( PF \) in \( G_b \) therefore we invoke \( W_{\phi,0}^b(p, \phi_{\text{F1}}) \) on every production \( p \) where \( h \) is \( \emptyset \) (since \( \phi_{\text{F1}} \) is a closed formula) and \( h \) maps the interface nodes of \( p \). In contrast to our previous example this time \( W_{\phi,0}^b(p, \phi_{\text{F1}}) = \phi_{\text{F1}} \) for all \( p \). We omit the workings as \( W_{\phi,0}^b(p, \phi_{\text{F1}}) \) is computed in a similar fashion as in Example 10 and § 8.1.

The next step requires that we check whether the graph \( G_b \setminus \sigma(EL) \) satisfies any of the weakest pre-conditions computed. Note that, even though \( W_{\phi,0}^b(p, \phi_{\text{F1}}) = \phi_{\text{F1}} \) for all \( p \) we are now checking whether \( G_b \setminus \sigma(EL) \models \phi_{\text{inv}} \). \( G_b \setminus \sigma(EL) \) is different depending on the production we are applying as well as the edge we are applying the production to.

For this specific example it is possible to fix the architectural style of \( G_b \) by applying initially the production \text{retryBooking} on the edge \( f_1 : F1 \) and then apply \text{paymentFailure} on the edge \( pf : PF \). Using now our methodology we know that since \( W_{\phi,0}^b(p, \phi_{\text{F1}}) = \phi_{\text{F1}} \) for all productions then \( W_{\phi,0}^b(p, W_{\phi,0}^b(p, \phi_{\text{F1}})) \) will also be equal to \( \phi_{\text{F1}} \) therefore we are only iterating step 3 replacing \( G_b \) with \( G_b \setminus \sigma(EL) \) where \( EL \) refers to the edge we are applying the production to. In this case \( G_b \) without edge \( f_1 : F1 \) returning \( G_b' \) and latter \( G_b'' \) without \( pf : PF \) to obtain \( G_b''' \). Note that we have to repeating step 3 because \( G_b \not\models \phi_{\text{F1}} \). We stop the methodology once we reach the point where \( G_b'''' \models \phi_{\text{F1}} \). Now using Theorem 1 we can assume that if \( G_b'''' \models W_{\phi,0}^b(p, \phi_{\text{F1}}) \) then the obtained graph will satisfy \( \phi_{\text{F1}} \).

9. Related work

Formal approaches based on architectural styles to control architectural reconfigurations have been proposed, among other, in [6, 7, 8, 3]. In those proposals reconfigurations are typically applied uniformly across the design. For instance, in [8, 3] graph grammars and hyper-edge replacements are used to
represent styles in terms of graph configurations freely generated by some productions (and it is not easy to specify conditions to extract subsets of such graph-languages).

Our work mitigates this effect by means of asserted productions that provide a finer control on the applicability conditions as done in other graph-transformation approaches. For instance, our approach is similar to the one in [9] where graph programs are extended to programs over high-level rules with application conditions; on such programs weakest pre-conditions can be defined automatically. Nevertheless, [9] aims at verifying computational properties of systems rather than architectural ones and does that in a different way only after generating the various state systems. In [10] constraints on the architecture are used to guarantee invariants of systems. More precisely, re-configurations can occur only if such constraints are not violated. This is not always realistic in open systems, therefore they do not impose limitations on run-time reconfigurations and search for new reconfigurations that can lead the system in a desired state.

In [11] an assume-guarantee mechanism is adopted to provide a learning algorithm which provides an assumption satisfying a sufficient condition in order for the component to guarantee the given invariant. This is achieved by model checking every component of the system against an invariant. This is similar to the weakest pre-condition we present in this paper but instead of computing the weakest assumption for every component of the system we compute the weakest pre-condition for every design production. We can later use our algorithm for applying the methodology described in § 7 for identifying the possible design production(s) (if any) to aid in fixing the architectural violation of the system.

In [12] the authors present an approach for designing safe systems by inspecting whether certain reconfigurations can lead to invalid graphs that represent invalid systems. This is achieved by verifying that the backward application of reconfigurations to a forbidden graph pattern cannot lead to a graph pattern representing a safe system (a set of forbidden graph patterns model an invariant). This method can provide a safe system in the sense that it cannot lead to a state that violates a structural invariant by the use of reconfigurations but it is very complex to handle unexpected system failures.

In [13] self-healing systems are modelled by specifying different types of rules; for the ideal system behaviour, for different predictable failures and for fixing the different failures identified earlier. This approach is different to what we propose in this paper as they design the rules according to the misbehaviour they expect at run time and do not necessarily handle unexpected failures or changes of the system.

Different approaches to specify self-managing systems are surveyed in [14]. The authors group the different approaches according to their ability to select different reconfigurations that should occur to re-establish a correct state. They present three type of selections namely, called pre-defined selection (a re-configuration is chosen prior to the execution based on a pre-defined selection), constrained selection from a pre-defined set (a reconfiguration designed for the given situation is chosen) and unconstrained selection (unconstrained choice re-
garding the appropriate change to make). All the approaches presented in the
survey lie in either of the former two categories and according to [14], none of the
approaches surveyed falls in the unconstrained selection category. Our approach
does not lie neither in the pre-defined nor in constrained selection categories.
It is not clear to us if our approach can be considered an unconstrained selec-
tion. In fact, we do not choose the reconfigurations to apply according to the
misbehaviour expected at run time. Instead we use our weakest pre-condition
algorithm to identify which of the existing configurations (not designed for the
specific violation) can re-establish the architectural style of our system. We re-
mark that most of the rules given at design time typically are meant to specify
the architectural style of a system, not its misbehaviour (for instance, in ADR
this might be addressed with reconfiguration rules rather than productions).
However, even if some productions were introduced to tackle (or prevent) some
misbehaviour, our approach enables such rules to be used also for unexpected
violations.

10. Conclusion and future work

We introduced a methodology inspired by Design by Contract (DbC) [15] to
guarantee properties of architectural designs. Technically this is achieved by (i)
equipping ADR with a logic tailored to express such properties and (ii) devising
an algorithm to compute weakest pre-conditions for ADR productions.

Albeit very simple, our logic can express rather interesting properties (cf.
Example 6). It allows us to improve the expressiveness of ADR and to specify
interesting properties exploiting the 'hierarchical nature' of ADR graphs. This
paper is a first step in the exploration of the use of DbC in architectural style
reconfigurations.

Using our methodology we can fix architecturally our graphs, provided that
we have the appropriate productions to do this. Our improved methodology
aids in recovering system violations of the architectural style, by recursively
computing the weakest pre-condition computed by the algorithm until a possible
sequence of reconfigurations is identified to re-establish the architectural style
of the system. More precisely, one can compute a sequence of productions by
iterating the methodology steps 3 and 4 in § 7 on the weakest pre-condition
obtained at every “round” (starting from the invariant) until either false or a
valid style is reached. We note that this opens other interesting questions. For
example, when different sequences of productions are found, one could devise
criteria to order them, or else to try to find criteria for good or best strategies.
Generalising our idea for computing ‘strategies’ based on many productions to
recover failures could be a very interesting future direction.

We expect such research to lead to extensions of the logic and also like stated
earlier extensions to the methodology to be able to handle more complex viola-
tions that might require more design productions to fix a system’s architecture.
References


Appendix A. Proofs for Theorem 1

In the proof of Theorem 1 we use the following observation.

Observation 1. By Definition 12, the algorithm computes the pre-condition of a quantified formula by considering every “reasonable” assignment of the quantified variables. More precisely, if the post-condition is a quantification $\forall D(\hat{x}).\phi$ (resp. $\exists D(\hat{x}).\phi$), the computed weakest pre-condition requires $\varphi'$ to be satisfied under all (resp. some) assignments of each variable in $\hat{x}$ to nodes in $R$ or to a fixed set of nodes $v_1, \ldots, v_n$ outside $R$. The identity of $v_1, \ldots, v_n$ is immaterial, in fact, the crucial point is that they refer to nodes outside $R$ (i.e., as many as the variables in $\hat{x}$).

For the sake of readability, we repeat the statement of Theorem 1.

Theorem 1. Let $p = (L, R, \delta)$ be a production, $\varphi \in \mathcal{L}$, $h : \text{fv}(\varphi) \to V_R$ be injective, $\tilde{h} : \mathcal{Z} \to V_L$ be a bijection where $\mathcal{Z} \subseteq \mathcal{V}$ disjoint from the variables in $\varphi$, $\mathcal{E}$ be an environment compatible with $h$, and $\pi$ be the asserted production $\{W_{h,\mathcal{E}}^\tilde{h}(p, \varphi), \tilde{h}\} p \{\varphi, h\}$. For any ADR graph $G$ and the morphism $\sigma$ from $L$ to $G$, if $G\setminus\sigma(E_L) \models_\mathcal{E} W_{h,\mathcal{E}}^\tilde{h}(p, \varphi)$ then $\pi(G, \sigma) \models_\mathcal{E} \mathcal{C}(\varphi)$. 

Proof. The proof is by induction on the structure of the post-condition $\varphi$.

- $\varphi$ is $\top$: we have $W_{h,\mathcal{E}}^\tilde{h}(p, \varphi) = \top$ and therefore the thesis follows trivially, because any graph satisfies $\top$.

- $\varphi$ is $\bot$: we have $W_{h,\mathcal{E}}^\tilde{h}(p, \varphi) = \bot$ and, since no graph satisfies $\bot$, the implication of the statement vacuously hold.

- $\varphi$ is $\varphi_1 \land \varphi_2$: by definition, $W_{h,\mathcal{E}}^\tilde{h}(p, \varphi) = \psi_1 \land \psi_2$ where $\psi_1 = W_{h,\mathcal{E}}^\tilde{h}(p, \varphi_1)$ and $\psi_2 = W_{h,\mathcal{E}}^\tilde{h}(p, \varphi_2)$. By inductive hypothesis, $G\setminus\sigma(E_L) \models_\mathcal{E} \psi_i$ implies $\pi(G, \sigma) \models_\mathcal{E} \mathcal{C}(\varphi_i)$ for $j = 1, 2$. The thesis follows by the definition of $\models$ (cf. Definition 7).

- $\varphi$ is $\forall D(\hat{x}).\varphi'$: let $d(\hat{u}) \in \pi(G, \sigma)$ such that $\tau_{\pi(G, \sigma)}(d) = D$ (i.e., the type of $d$ is $D$) and $\psi = W_{h,\mathcal{E}}^\tilde{h}(p, \varphi')$. By induction, $G\setminus\sigma(E_L) \models_\mathcal{E} \psi$ implies that $\pi(G, \sigma) \models_\mathcal{E} \mathcal{C}(\varphi')$.

The thesis follows by observing that there are only finitely many edges $d(\hat{u})$ satisfying the property above and by the fact that, by Definition 12, $W_{h,\mathcal{E}}^\tilde{h}(p, \varphi)$ is obtained by the conjunction of

$$
wd_{\mathcal{E}}^{\tilde{h}}(\varphi) = \bigwedge_{\hat{u} \text{ on } R \cdot D} wd_{\mathcal{E}}^{\tilde{h}}(\varphi')
$$

$$
wp_{\mathcal{E}, h}^{D, \tilde{h}}(\varphi) = \bigwedge_{\hat{u} \text{ on } R \cdot D} \forall D(\hat{x}). wp_{\mathcal{E}, D}^{\tilde{h}}(\varphi')
$$

for some $\mathcal{E}$.
$\varphi$ is $\exists D(\bar{x}).\varphi'$: By contradiction, suppose that $\pi(G, \sigma) \not\models_{E} C(\varphi')$ for any tuple of nodes $\bar{u}$ such that $d(\bar{u}) \in \pi(G, \sigma)$ with $\tau_{\pi(G, \sigma)}(d) = D$ (i.e., the type of $d$ is $D$). Fix such a tuple $\bar{u}$, assign $\bar{u}$ in $E$ to $\bar{x}$ and let $\psi' = W_{h,E}^{h}(p, \varphi')$. By induction, we have that $G \setminus \sigma(E_L) \models_{E} \psi$ implies that $\pi(G, \sigma) \models_{E} C(\varphi')$ which yields a contradiction. Hence, there should exist an edge $d(\bar{u}) \in \pi(G, \sigma)$ such that $\pi(G, \sigma) \models_{E} C(\varphi')$ The thesis follows by observing that, by Definition 12, in this case $W_{h,E}^{h}(p, \varphi)$ returns

$$wd_{E}^{p,\psi}(\varphi) = \bigvee_{\bar{u}} \text{ on } R \cdot D \; wd_{E'}^{p,\psi}(\varphi')$$

$$wp_{E',h}^{p,\bar{h}}(\varphi) = \bigvee_{\bar{u}} \text{ on } R \cdot D \; (\exists D(\bar{x}).wp_{E',h[x\mapsto(\exists,D,\bar{u})]}^{p,\bar{h}}(\varphi')) \lor wd_{E'}^{p,\psi}(\varphi'))$$

for some $E$. Observe that $wd_{E}^{p,\psi}(\varphi')$ returns $\top$ when the RHS of $p$ satisfies the post-condition $\varphi$. For an assignment in which this is the case, the disjunct is just $\top$ since, the graph resulting from the application of $p$ would guarantee the post-condition due to the instantiation of the RHS of $p$ under such assignment. In all the other cases, the disjunct is $\bigvee_{\bar{u}} \text{ on } R \cdot D \; (\exists D(\bar{x}).wp_{E',h[x\mapsto(\exists,D,\bar{u})]}^{p,\bar{h}}(\varphi'))$. Hence, the pre-condition for an existential quantification requires the existence of suitable edges in the graph $p$ is applied to whenever an assignment does not make the RHS of $p$ to satisfy the post-condition.

$\varphi$ is $x_1 = x_2$: let $E$ be an environment as in Definition 12 and recall that variables $x_1$ and $x_2$ are indexed according to the order in which they are quantified in $\varphi$ (if at all). By definition the algorithm invokes the auxiliary map of Definition 11 and returns

$$eq_{p,x_1 = x_2}^{\text{noEdge}(D_2),\text{noEdge}(D_1,D_2)}(E)$$

(A.1)

Hence it suffices to consider all the cases of the auxiliary map. In the following cases, recall also Observation 1 where, depending on the quantification, the algorithm will return the conjunction or disjunction of all the possible assignments of the quantified variables.

- If (A.1) is $\bot$ then the thesis follows trivially.
- If (A.1) is $\top$ then one of the following holds:
  - both $x_1$ and $x_2$ are existentially quantified and mapped to the same internal node of $R$
  - one of the variables is existentially quantified and mapped to an internal node of $R$ and the other is a free variable mapped to the same node
  - $x_1$ is universally quantified, $x_2$ is existentially quantified and both of them are mapped to the same internal node in $R$
– both variables are free variables mapped to the same internal node of \( R \).

Therefore, any graph obtained by applying the production will have edges of the appropriate type sharing the node assigned to \( x_1 \) and \( x_2 \) since such edges and nodes are introduced by the RHS \( R \) of the production \( p \).

- If (A.1) is \( \text{noEdge}(D_2) \) then one of the following cases holds:

  - \( \mathcal{E}^{(1)}(x_1) \uparrow, \mathcal{E}(x_2) \equiv \forall D_2 R^o \), and the two variables are mapped to the same internal node. If \( G \setminus \sigma(E_L) \) has an edge of type \( D_2 \), then \( G \setminus \sigma(E_L) \not\equiv \text{noEdge}(D_2) \) and the thesis follows vacuously.

  - \( \mathcal{E}(x_1) \equiv \forall D_1 R^o \) and \( \mathcal{E}(x_2) \equiv \forall D_2 R^o \). In this case, \( L \), the LHS of \( p \), has a type different from \( D_2 \) (otherwise the morphism \( \sigma \) could not exist). Hence, if \( G \) has an edge of type \( D_2 \) then \( G \setminus \sigma(E_L) \not\equiv \text{noEdge}(D_2) \) and the thesis follows vacuously.

- (A.1) is \( \text{noEdge}(D_1, D_2) \) when \( \mathcal{E}(x_1) \equiv \forall D_1 n, \mathcal{E}(x_2) \equiv \forall D_2 n \) and
\( n \in R^c \). Since edges of type \( D_1 \) and \( D_2 \) are introduced by \( R \) as specified by the mapping and \( n \) is a fresh internal node then there can be no more edges of these types in \( G' \setminus \sigma(E_L) \). If \( G' \setminus \sigma(E_L) \) has an edge of type \( D_1 \) or \( D_2 \) then \( G' \setminus \sigma(E_L) \not\models \text{noEdge}(D_1, D_2) \) and the thesis follows vacuously. If \( G' \setminus \sigma(E_L) \models \text{noEdge}(D_1, D_2) \), then the only edges of type \( D_1 \) and \( D_2 \) in \( \pi(G, \sigma) \) are those introduced by the RHS of \( p \) and if all of them are attached to \( n \) as checked by the conjunction on every possible mappings \( \hat{u} \) on \( R \cdot D_1 \) and \( \hat{u} \) on \( R \cdot D_2 \) then by Observation 1, \( \pi(G, \sigma) \models \varphi \) otherwise the pre-condition would be \( \bot \), contrary to the hypothesis.

- (A.1) is \( x_1 = x_2 \) in the following cases:

  1. \( \mathcal{E}(x_1) \triangleq \exists D_1 \ n, \mathcal{E}(x_2) \triangleq \exists D_2 \ n' \) and \( n \notin R^c \) or \( n' \notin R^c \). If \( G' \setminus \sigma(E_L) \not\models \mathcal{E}(x_1) \triangleq x_1 = x_2 \) (i.e., \( x_1 \) and \( x_2 \) are assigned to different nodes in \( E \)) then the thesis follows vacuously. If \( G' \setminus \sigma(E_L) \models \mathcal{E}(x_1) \triangleq x_1 = x_2 \) then, due to the existential quantifications, \( \pi(G, \sigma) \models \mathcal{E}(x_1) \triangleq x_1 = x_2 \) holds if there is an assignment mapping \( x_1 \) and \( x_2 \) to the same node (by Observation 1). Since \( \pi(G, \sigma) \) is the graph obtained by the union of \( G' \setminus \sigma(E_L) \) and \( R \) then \( \pi(G, \sigma) \) "inherits" from \( G' \setminus \sigma(E_L) \) the node assigned to \( x_1 \) and \( x_2 \) and hence satisfies the invariant.

  2. \( \mathcal{E}(x_1) \triangleq \forall D_1 \ n, \mathcal{E}(x_2) \triangleq \forall D_2 \ n' \) and \( n, n' \notin R^c \). If there is no node assigned to \( x_1 \) in \( G' \setminus \sigma(E_L) \) such that every assignment on \( x_2 \) is equal to \( x_1 \) then \( G' \setminus \sigma(E_L) \not\models \mathcal{E}(x_1) \triangleq x_1 = x_2 \) and the thesis follows vacuously. If \( G' \setminus \sigma(E_L) \models \mathcal{E}(x_1) \triangleq x_1 = x_2 \), then \( \pi(G, \sigma) \models \mathcal{E}(x_1) \triangleq x_1 = x_2 \) since there are no other edges of type \( D_2 \) in \( R \); in fact, the conjunction on every possible mapping \( \hat{u} \) on \( R \cdot D_2 \) would not hold otherwise, by Observation 1 and therefore the computed pre-condition would have been \( \text{noEdge}(D_2) \) and not \( x_1 = x_2 \) as by hypothesis.

  3. \( \mathcal{E}(x_1) \triangleq \forall D_1 \ n, \mathcal{E}(x_2) \triangleq \forall D_2 \ n' \) and \( n \neq n' \). If \( n \neq n' \) then \( G' \setminus \sigma(E_L) \not\models \mathcal{E}(x_1) \triangleq x_1 = x_2 \) and the thesis follows vacuously. If \( G' \setminus \sigma(E_L) \models \mathcal{E}(x_1) \triangleq x_1 = x_2 \), then \( \pi(G, \sigma) \models \mathcal{E}(x_1) \triangleq x_1 = x_2 \) due to the fact that there are no other edges of type \( D_1 \) and \( D_2 \) in \( R \); in fact, the conjunction on every possible mapping \( \hat{u} \) on \( R \cdot D_1 \) and \( \hat{u} \) on \( R \cdot D_2 \) has to hold otherwise, by Observation 1, the computed pre-condition would have been \( \text{noEdge}(D_1, D_2) \) and not \( x_1 = x_2 \) as by hypothesis.

  4. \( \mathcal{E}(x_1) \triangleq \forall D_1 \ n, \mathcal{E}(x_2) \triangleq \exists D_2 \ n' \) and \( n \notin R^c \). If there is a node assigned to \( x_1 \) in \( G' \setminus \sigma(E_L) \) such that no assignment on \( x_2 \) is equal to \( x_1 \) then \( G' \setminus \sigma(E_L) \not\models \mathcal{E}(x_1) \triangleq x_1 = x_2 \) and the thesis follows vacuously. If \( G' \setminus \sigma(E_L) \models \mathcal{E}(x_1) \triangleq x_1 = x_2 \), then \( \pi(G, \sigma) \models \mathcal{E}(x_1) \triangleq x_1 = x_2 \) if there is a node assigned to \( x_1 \) in \( R \) not equal to any node assigned to \( x_2 \); in fact, the conjunction on every possible mapping \( \hat{u} \) on \( R \cdot D_1 \) and \( \hat{u} \) on \( R \cdot D_2 \) has to hold otherwise, by Observation 1, the computed pre-condition would have been \( \bot \)
and not $x_1 = x_2$ as by hypothesis.

$\varphi$ is $x_1 \neq x_2$: let $E$ be an environment as in Definition 12. By definition the algorithm invokes the auxiliary map of Definition 11 and returns

$$
\neg \phi_{\mathcal{P}, x_1 = x_2} (E)
$$

(A.2)

Hence it suffices to consider all the possible cases of the auxiliary map.

- If (A.2) is $\bot$ then the thesis follows trivially.
- If (A.2) is $\top$ when one of the following cases occur:
  - $\mathcal{E}(x_1) \equiv \exists D_1 \ R^o, \ \mathcal{E}^{(1)}(x_2) \uparrow$ and $\mathcal{E}^{(3)}(x_1) \neq \mathcal{E}^{(3)}(x_2)$ If the node assigned to $x_1$ by $\mathcal{E}^{(3)}$ is not equal to the free node of the post-condition mapped to $x_2$ then the invariant is satisfied regardless of $G \sigma(E_L)$ due to the existential quantification on $x_1$ and the algorithm returns $\top$. $G \sigma(E_L) \models \top$ and the obtained $\pi(G, \sigma) \models_{\mathcal{E}(3)} x_1 \neq x_2$ since the instance of $R$ that replaces $L$ in $G$ guarantees the existential quantification of the inequality.
  - If $\mathcal{E}(x_2) \equiv \forall D \ R^o, \ \mathcal{E}^{(1)}(x_2) \uparrow$ and $\mathcal{E}^{(3)}(x_1) \neq \mathcal{E}^{(3)}(x_2) \uparrow$. If every node in $R$ assigned to $x_1$ by $\mathcal{E}^{(3)}$ is not equal to the free node of the post-condition mapped to $x_2$ then the invariant is satisfied as long as there is no other edge of type $D$ in $G \sigma(E_L)$. $G \sigma(E_L) \models \top$ and the obtained $\pi(G, \sigma) \models_{\mathcal{E}(3)} x_1 \neq x_2$ if there are no other edges of type $D$ in $R$; in fact, the conjunction on every possible mapping $\bar{u}$ on $R \cdot D$ has to hold otherwise, by Observation 1, the computed pre-condition would have been $\text{noEdge}(D)$ and not $\top$ as by hypothesis.
  - As in the previous case if variable $x_1$ is universally quantified and this time mapped to either an interface node or an external node while $\mathcal{E}^{(1)}(x_2) \uparrow$ then $G \sigma(E_L) \models \top$ and the obtained $\pi(G, \sigma) \models_{\mathcal{E}(3)} x_1 \neq x_2$ if there is no other edge of type $D$ in $R$ attached to $x_2$: in fact, the conjunction on every possible mapping $\bar{u}$ on $R \cdot D$ has to hold otherwise, by Observation 1, the computed pre-condition would have been $\bot$ and not $\top$ as by hypothesis.
  - $\mathcal{E}(x_1) \equiv \exists D_1 \ R^o, \ \mathcal{E}(x_2) \equiv \forall D_2 \ n$ and $n \notin R^o$. Since the node assigned by $\mathcal{E}^{(3)}$ to $x_1$ is internal and to $x_2$ is external then they are disjoint and the algorithm returns $\top$ for this case. $G \sigma(E_L) \models \top$ and the obtained $\pi(G, \sigma) \models_{\mathcal{E}(3)} x_1 \neq x_2$ if there is no node in $R$ assigned to $x_2$ by $\mathcal{E}^{(3)}$ equal to the node assigned to $x_1$; in fact, the conjunction on every possible mapping $\bar{u}$ on $R \cdot D_2$ has to hold otherwise, by Observation 1, the computed pre-condition would have been $x_1 \neq x_2$ and not $\top$ as by hypothesis.
  - $\mathcal{E}(x_1) \equiv \forall \ n, \ \mathcal{E}(x_2) \equiv \exists \ n$ and $n \notin R$ then like the previous case $x_1$ and $x_2$ are distinct nodes and the algorithm returns $\top$.
for this case. \( G \setminus \sigma(E_L) \models \top \) and the obtained \( \pi(G, \sigma) \models \varepsilon(3) \) \( x_1 \neq x_2 \) if there is no node assigned to \( x_1 \) by \( \varepsilon(3) \) equal to the node assigned to \( x_2 \); in fact, the conjunction on every possible mapping \( \tilde{u} \) on \( R \cdot D_1 \) and \( \tilde{u} \) on \( R \cdot D_2 \) has to hold otherwise, by Observation 1, the computed pre-condition would have been \( x_1 \neq x_2 \) and not \( \top \) as by hypothesis.

- \( \varepsilon(x_1) \models \forall \_ R^\circ, \varepsilon(x_2) \models \exists \_ R^\circ \) and \( \varepsilon(3)(x_1) \neq \varepsilon(3)(x_2) \) then \( x_1 \) and \( x_2 \) are mapped by \( \varepsilon(3) \) to distinct nodes in this case and the algorithm returns \( \top \). \( G \setminus \sigma(E_L) \models \top \) and the obtained \( \pi(G, \sigma) \models \varepsilon(3) \) \( x_1 \neq x_2 \) if there are no other nodes assigned to \( x_1 \) and \( x_2 \) such that they are equal; in fact, the conjunction on every possible mapping \( \tilde{u} \) on \( R \cdot D_1 \) and \( \tilde{u} \) on \( R \cdot D_2 \) has to hold otherwise, by Observation 1, the computed pre-condition would have been \( \bot \) and not \( \top \) as by hypothesis.

- If \( \varepsilon(x_1) \models \exists \_ R \) and \( \varepsilon(x_2) \models \forall \_ R^\circ \) then since \( x_2 \) is a fresh internal node this means that it cannot be equal to the external node mapped to \( x_1 \) and in this case the algorithm returns \( \top \). \( G \setminus \sigma(E_L) \models \top \) and the obtained \( \pi(G, \sigma) \models \varepsilon(3) \) \( x_1 \neq x_2 \) if there is a node assigned to \( x_1 \) distinct to every node assigned to \( x_2 \) by \( \varepsilon(3) \); in fact, the conjunction on every possible mapping \( \tilde{u} \) on \( R \cdot D_1 \) and \( \tilde{u} \) on \( R \cdot D_2 \) has to hold otherwise, by Observation 1, the computed pre-condition would have been \( \bot \) and not \( \top \) as by hypothesis.

- If \( \varepsilon(x_1) \models \exists \_ R^\circ, \varepsilon(x_2) \models \exists \_ R^\circ \) and \( \varepsilon(3)(x_1) \neq \varepsilon(3)(x_2) \). If the node assigned to \( x_1 \) by \( \varepsilon(3) \) is not equal to the node assigned to \( x_2 \) then the invariant is satisfied regardless of \( G \setminus \sigma(E_L) \) due to the existential quantification on both \( x_1 \) and \( x_2 \) and therefore algorithm returns \( \top \). \( G \setminus \sigma(E_L) \models \top \) and the obtained \( \pi(G, \sigma) \models \varepsilon(3) \) \( x_1 \neq x_2 \) since the instance of \( R \) that replaces \( L \)
in $G$ guarantees the existential quantification of the inequality.

\[ \square \]

Appendix B. Proofs for Theorem 2

For the sake of readability, we repeat the statement of Theorem 2.

**Theorem 2.** Given $\varphi \in \mathcal{L}$, let $Z \subseteq V$ such that no variables in $\varphi$ is in $Z$ and $h : Z \rightarrow V_L$ be a bijection, then for any formula $\psi$ such that $\{ \psi, h \} \vdash \{ \varphi, h \}$ is a valid production for any graph $G$ then $\psi$ implies $\mathcal{W}_{h,E}^h(p, \varphi)$.

**Proof.**

$\mathcal{W}_{h,E}^h(p, \varphi)$ is $\top$: This case is trivial.

$\mathcal{W}_{h,E}^h(p, \varphi)$ is $\bot$: when either $\varphi$ is $\bot$ or when $\varphi$ is $x_1 = x_2$ for any case cond$_\bot$ in Definition 11. Recall that $\sigma$ is the morphism from the LHS of a production to the graph $G$ and that $\pi(G, \sigma)$ refers to the graph obtained after the application of the production.

$\varphi$ is $\bot$: we show that $\psi$ is equivalent to $\bot$; assume this is not the case. Then there is $G \sigma(\mathcal{E}_L) \models_h \psi$ and, by the validity of $\{ \psi, h \} \vdash \{ \varphi, h \}$ follows that $\pi(G, \sigma) \not\models_h \varphi$ that is $G \sigma(\mathcal{E}_L) \models_h \bot$, which is absurd.

$\varphi$ is $x_1 = x_2$: for any of the cases cond$_\bot$ in Definition 11 we show that $\psi$ is equivalent to $\bot$; assume this is not the case. By the following case analysis we observe that for every case in cond$_\bot$ since $\mathcal{W}_{h,E}^h(p, \varphi) = \bot$ there is a graph $G$ s.t. $\pi(G, \sigma) \not\models_h \varphi$ and hence the production would not be valid if $G \sigma(\mathcal{E}_L) \models_h \psi$. The first four cases bellow show that since the environment $\mathcal{E}$ is compatible to $h$ and $\pi(G, \sigma) \not\models_h x_1 = x_2$ then the production would not be valid if $G \sigma(\mathcal{E}_L) \models_h \psi$. For the rest of the cases we assume the conjunction on every possible mapping $\bar{u}$ on $R \cdot D$ returns $\bot$ and therefore as in the other cases the fact that $\mathcal{E}$ is compatible to $h$ and that $\pi(G, \sigma) \not\models_h x_1 = x_2$ implies that if $G \sigma(\mathcal{E}_L) \models_h \psi$ then the production cannot be valid.

1. If $\mathcal{E}(x_1) \models \exists \bar{D}_1 \in \mathcal{R}^c$, $\mathcal{E}^{(1)}(x_2) \uparrow$ and $\mathcal{E}^{(3)}(x_1) \neq \mathcal{E}^{(3)}(x_2)$ then $\pi(G, \sigma) \not\models_h x_1 = x_2$.
2. If $\mathcal{E}(x_1) \models \forall \bar{D}_1 \in \mathcal{R}^c$, $\mathcal{E}^{(1)}(x_2) \uparrow$ and $\mathcal{E}^{(3)}(x_1) \neq \mathcal{E}^{(3)}(x_2)$ then $\pi(G, \sigma) \not\models_h x_1 = x_2$.
3. If $\mathcal{E}(x_1) \models \exists n \in \mathcal{R}^c$, $\mathcal{E}^{(1)}(x_2) \uparrow$ then $\pi(G, \sigma) \not\models_h x_1 = x_2$.
4. If $\mathcal{E}(x_1) \models \exists n \in \mathcal{R}^c$, $\mathcal{E}^{(1)}(x_2) \uparrow$ and $n \not\in \mathcal{R}^c$. then $\pi(G, \sigma) \not\models_h x_1 = x_2$.
5. If $\mathcal{E}(x_1) \models \exists n \in \mathcal{R}^c$, $\mathcal{E}^{(2)}(x_2) \models \exists n \not\in \mathcal{R}^c$ then since $x_1$ is an internal node and $x_2$ an external node this implies they are distinct nodes and $\pi(G, \sigma) \not\models_h x_1 = x_2$.
6. If $\mathcal{E}(x_1) \models \exists n \in \mathcal{R}^c$, $\mathcal{E}^{(2)}(x_2) \models \exists n \not\in \mathcal{R}^c$ and $\mathcal{E}^{(3)}(x_1) \neq \mathcal{E}^{(3)}(x_2)$ then $\pi(G, \sigma) \not\models_h x_1 = x_2$.
7. If \( \mathcal{E}(x_1) \equiv \forall \, D_1 \, \mathit{Im}(i) \), \( \mathcal{E}(x_2) \equiv \exists \, D_2 \) \( \bar{\mathit{R}} \) and \( D_1 \in R \) this means that an edge of type \( D_1 \) is in \( R \) and the mapped node to \( x_1 \) is not equal to \( x_2 \) and therefore \( \pi(G, \sigma) \not\ni \bar{h} \, x_1 = x_2 \).

8. If \( \mathcal{E}(x_1) \equiv \exists \, \bar{\mathit{R}} \) and \( \mathcal{E}(x_2) \equiv \forall \, \mathit{R}^o \) then \( \pi(G, \sigma) \not\ni \bar{h} \, x_1 = x_2 \) since \( x_2 \) is mapped to an internal node and \( x_1 \) to an external node.

9. If \( \mathcal{E}(x_1) \equiv \exists \, \mathit{R}^o \), \( \mathcal{E}(x_2) \equiv \forall \, \mathit{R}^o \) and \( \mathcal{E}^{(3)}(x_1) \neq \mathcal{E}^{(3)}(x_2) \) then \( \pi(G, \sigma) \not\ni \bar{h} \, x_1 = x_2 \).

10. \( \mathcal{E}(x_1) \equiv \exists \, D_1 \, \mathit{R}^o \), \( \mathcal{E}(x_2) \equiv \forall \, D_2 \) \( n \) and \( n \notin \mathit{R}^o \) then \( \pi(G, \sigma) \not\ni \bar{h} \, x_1 = x_2 \).

\( \mathcal{W}^h_{\bar{h}, \bar{e}}(p, \varphi) \) is \( \mathit{noEdge}(D_1, D_2) \): If \( \psi \) does not imply \( \mathit{noEdge}(D_1, D_2) \), there should be a graph \( G \) such that \( G, \sigma(E_L) \not\ni \bar{h} \, \psi \land \lnot \mathit{noEdge}(D_1, D_2) \). This yields a contradiction since by the case analysis provided in the proofs for Theorem 1 when \( G, \sigma(E_L) \not\ni \bar{h} \) \( \mathit{noEdge}(D_1, D_2) \) then \( \pi(G, \sigma) \not\ni \bar{h} \, \varphi \) under this hypothesis and hence the production would not be valid if \( G, \sigma(E_L) \not\vdash \psi \).

\( \mathcal{W}^h_{\bar{h}, \bar{e}}(p, \varphi) \) is \( \mathit{noEdge}(D_2) \): If \( \psi \) does not imply \( \mathit{noEdge}(D_2) \), there should be a graph \( G \) such that \( G, \sigma(E_L) \not\ni \bar{h} \, \psi \land \lnot \mathit{noEdge}(D_2) \). This yields a contradiction since by the case analysis provided in the proofs for Theorem 1 when \( G, \sigma(E_L) \not\ni \bar{h} \) \( \mathit{noEdge}(D_2) \) then \( \pi(G, \sigma) \not\ni \bar{h} \, \varphi \) under this hypothesis and hence the production would not be valid if \( G, \sigma(E_L) \not\vdash \psi \).

\( \mathcal{W}^h_{\bar{h}, \bar{e}}(p, \varphi) \) is \( x_1 = x_2 \): This case is analogous to the previous one. If \( \psi \) does not imply \( x_1 = x_2 \), there should be a graph \( G \) such that \( G, \sigma(E_L) \not\ni \bar{h} \, \psi \land \lnot x_1 = x_2 \). This yields a contradiction since by the case analysis provided in the proofs for Theorem 1 when \( G, \sigma(E_L) \not\ni \bar{h} \) \( x_1 = x_2 \) then \( \pi(G, \sigma) \not\ni \bar{h} \, \varphi \) under this hypothesis and hence the production would not be valid if \( G, \sigma(E_L) \not\vdash \psi \).

\( \mathcal{W}^h_{\bar{h}, \bar{e}}(p, \varphi) \) is \( x_1 \neq x_2 \): This case is analogous to the previous one.

\( \mathcal{W}^h_{\bar{h}, \bar{e}}(p, \varphi) \) is \( \varphi \land \varphi'; \): By the inductive hypothesis, \( \varphi \) implies both \( \varphi_1 \) and \( \varphi_2 \).

Hence, \( \varphi \) implies \( \varphi_1 \land \varphi_2 = \mathcal{W}^h_{\bar{h}, \bar{e}}(p, \varphi_1) \land \mathcal{W}^h_{\bar{h}, \bar{e}}(p, \varphi_2) = \mathcal{W}^h_{\bar{h}, \bar{e}}(p, \varphi) \), where the last equality holds by definition of our algorithm.

\( \mathcal{W}^h_{\bar{h}, \bar{e}}(p, \varphi) \) is \( \forall D(\bar{x}), \varphi' \): By contradiction, let \( \varphi = \forall D(\bar{x}), \varphi' \) and recall that in this case it should be \( \varphi = \forall D(\bar{x}), \varphi'' \) for some \( \varphi'' \in L \). If \( \psi \) does not imply \( \forall D(\bar{x}), \varphi' \), there should be a graph \( G \) and a mapping \( h' : \mathfrak{f}(\psi \land \varphi) \rightarrow V_G \) such that \( G \not\vdash h' \, \psi \land \lnot \forall D(\bar{x}), \varphi' \), namely there is \( e(\bar{u}) \in G \) such that \( e \) is of type \( D \) and \( G \not\vdash h'(\bar{x} \rightarrow \bar{a}) \, \psi \land \lnot \varphi' \). This yields a contradiction since by inductive hypothesis if \( G, \sigma(E_L) \not\ni \bar{h} \, \varphi' \) then \( \pi(G, \sigma) \not\ni \varphi \) and if \( G, \sigma(E_L) \not\vdash h'(\bar{x} \rightarrow \bar{a}) \, \varphi' \) then \( \psi \rightarrow \varphi' \).

\( \mathcal{W}^h_{\bar{h}, \bar{e}}(p, \varphi) \) is \( \exists D(\bar{x}), \varphi' \): This case is analogous to the previous one.

\( \square \)