Strategies for consistency checking based on unification

H. Bowman, E.A. Boiten, J. Derrick, M.W.A. Steen

Computing Laboratory, University of Kent, Canterbury CT2 7NF, UK

Communicated by M.C. Gaudel; received 5 April 1996; received in revised form 10 March 1997

Abstract

There is increasing interest in models of system development which use Multiple Viewpoints. Each viewpoint offers a different perspective on the target system and system development involves parallel refinement of the multiple views. Multiple viewpoints though, prompt the issue of consistency between viewpoints. This paper describes an interpretation of consistency which is general enough to meet the requirements of consistency for very general viewpoints models. Furthermore, the paper investigates strategies for checking this consistency definition. Particular emphasis is placed on mechanisms to obtain global consistency (between an arbitrary number of viewpoints) from a series of binary consistency checks. The consistency checking strategies we develop are illustrated using the formal description technique LOTOS. © 1999 Elsevier Science B.V. All rights reserved.

Keywords: Viewpoints; Consistency; Formal description techniques; LOTOS

1. Introduction

There has been significant recent interest in using viewpoints in system development. In such modelling, each viewpoint offers a different perspective on the target system and system development involves parallel refinement of the multiple views. Notable proponents of viewpoints modelling include [18, 28, 1, 19]. All these approaches prompt the central issue of viewpoint consistency, i.e. how to check that multiple specifications of the system do not conflict with one another and are "in some sense" consistent. Our perspective on consistency is tinged by the particular application of viewpoints that our work has been targetted at, viz. the viewpoints model defined in the ISO/ITU Open Distributed Processing (ODP) standardisation framework. ODP defines a generic framework to support the open interworking of distributed systems components. A central tenet of ODP is the use of viewpoints in order to decompose the task of specifying
distributed systems. ODP supports five viewpoints: Enterprise, Information, Computational, Engineering and Technology. In contrast to many other viewpoint models, ODP viewpoints are predefined and in this sense static, i.e. new viewpoints cannot be added. Each of the viewpoints has a specific purpose and is targeted at a particular class of specification. A complete ODP specification should contain a description of the system from each of the defined viewpoints. In addition, formal description techniques (FDTs) are variously applicable to the specification requirements of the different viewpoints. For example, Z [27] is being proposed for the information viewpoint and LOTOS [3] for the computational viewpoint.

Fig. 1 [14] depicts the relationships that are involved in relating ODP viewpoints. Development yields a specification that defines the system being described more closely. The term development embraces many mechanisms for evolving descriptions towards implementations, one of which is refinement. Because all five viewpoint specifications will eventually be realised by one system, there must be a way to combine specifications from different viewpoints during development; this is known as unification. For specifications in different FDTs to be combined or unified, a translation mechanism is needed to transform a specification in one language to a specification in another language. Consistency is a relation between groups of specifications.

In our work on consistency we distinguish between intra and inter language consistency checking. Intra language consistency considers how multiple specifications in the same language can be shown to be consistent, while inter language consistency considers relations between specifications in different FDTs.

The latter issue is a significantly more demanding topic than the former and although some techniques which are specific to particular pairs of languages do exist, e.g. between LOTOS and Z [16], no formal framework for inter language consistency exists.
Thus, the formal theory presented in this paper largely restricts itself to intra language consistency.

In order to inform the definition of consistency we choose it is worth considering what we require of such a definition. We offer the following list as a set of general requirements. The consistency definition we seek must

- Be applicable *intra language* for many different FDTs, e.g. must make sense between two Z specifications and also between two LOTOS specifications.
- Be applicable *inter language* between different FDTs, e.g. relate a Z specification to a LOTOS specification.
- Support different classes of consistency check. There are many different forms of consistency and the appropriate check to apply depends on the viewpoint specifications being considered and the relationship between these viewpoints [7]. For example, it would be inappropriate to check two specifications which express exactly corresponding functionality with the same notion of consistency that is applicable to checking consistency between specifications which extend each other's functionality.
- Support global consistency. Much of the work, to date, on consistency has only considered the case of two viewpoints (what we will call binary consistency); for full generality we need any arbitrary number of viewpoints greater than zero.
- Allow different viewpoints to relate to the target system in different ways. Thus, not only are there different forms of consistency check, but within a consistency check, specifications are related in different ways. For example, the enterprise specification is likely to express global requirements, while the computational specification defines an interaction model. Thus, the relationship between the system being developed and the enterprise specification is very different from the relationship of the system to the computational specification.

The last point prompts our work on, so-called, unbalanced consistency in which each viewpoint is potentially related to the system under development by a different development relation. For example, the enterprise viewpoint may be related by a logical satisfaction relation while the computational viewpoint may be related by a behavioural conformance relation. Note also that unbalanced consistency is needed to support inter language consistency. This aspect of our work represents a significant departure from existing theoretical work on relating partial specifications, e.g. [1, 31], which has typically looked at, what we call, balanced consistency.

An important approach which embraces a number of formalisms is institutions [20]. Ref. [9] briefly compares institutions to our model. The comparison gives a number of reasons why we have developed a new approach, the most important of which perhaps is that the theory of institutions generalises logical frameworks and hence uses satisfaction as its core correctness relation. However, our framework is parameterised on the choice of development, which could be one of many relations.

We have considered viewpoint consistency for ODP in a number of papers [5, 7, 8, 17, 29] and most fully in [2]. In particular, we have located a general definition of consistency and investigated properties of the definition. However, the issue of strategies for checking consistency remains open. In response, this paper considers,
in general terms, strategies for checking consistency according to our basic definition. The main contribution of the paper is to investigate how to obtain global consistency incrementally from a series of, probably binary, consistency checks. The paper will highlight a number of different classes of consistency checking. These vary from the very poorly behaved, where, realistically, it is impossible to check global consistency incrementally, to the very well behaved, where all groups of specifications are trivially consistent. Throughout we will illustrate the consistency checking problem using LOTOS and Z; although, particular emphasis will be placed on LOTOS.

The paper begins by reviewing our interpretation of consistency in Section 2 and proving some simple properties of the definition. Then in Section 3 we present background on LOTOS and some of its development relations. Section 4 highlights basic strategies for checking global consistency. In particular, two classes of consistency checking are identified: when a unique minimally developed unification does not exist and when such a unique minimal development (i.e. a least development) can be found. These two classes are considered separately in Sections 5 and 6, respectively. Then Section 7 discusses another restricted class of consistency checking, i.e. balanced consistency checking. Concluding remarks are presented in Section 8.

2. A general interpretation of consistency

We will give general definitions of the consistency checking relationships: consistency, both intra and inter language, and unification. First though we present the notation that we will work with. Importantly, this notation reflects the search for a general interpretation of consistency by defining very general notational conventions.

Notation. We begin by assuming a set DES of formal descriptions, which contains both formal specifications in languages such as LOTOS and Z and semantic descriptions in notations such as labelled transition systems [26] and trace/refusal semantics [10].

We assume a set DEV \( \subseteq \mathcal{P}(DES \times DES) \) of development relations. These are written \( dv \) and if \( X \ dv \ X' \) then, in some sense, \( X \) is a valid development of \( X' \). Our concept of a development relation generalises all notions of evolving a formal description towards an implementation and thus embraces the many such notions that have been proposed. In particular, DEV contains refinement relations, equivalences and relations which can broadly be classed as implementation relations. These different classes of development are best distinguished by their basic properties. Refinement is typically reflexive and transitive (i.e. a preorder); equivalences are reflexive, symmetric and transitive; and implementation relations are only reflexive.

To illustrate these concepts, the following examples of development relations can be highlighted and many more can be found in the literature.

- Refinement relations. In the process algebra domain, failures divergences refinement [12, 13] is a well-known preorder. The relation is defined between CSP processes and characterises refinement as reducing non-determinism. Related refinement relations exist for LOTOS and will be discussed in Section 3. Another example
is refinement in the state-based world, which is often based upon the existence of a simulation [21]. For example, refinement between two Z specifications is verified by showing that the concrete specification is a simulation of the abstract specification. An extensive discussion on simulation-based refinement in Z is given in [30].

- **Equivalence relations.** Important examples of equivalences include the bisimulation equivalences [26] which characterise when two process algebra specifications can be viewed to be indistinguishable to an external observer. Hennessy and De Nicola testing equivalence characterises a (weaker) equivalence that reflects an alternative view of what is indistinguishable to external observers. We will consider a testing equivalence in Section 3 that is closely related to the Hennessy/De Nicola equivalence.

- **Implementation relations.** A good example of an implementation relation [24] is conf [10] which we will consider in Section 3. This represents one perspective on the conformance testing process, which it is argued in [10] is not necessarily a transitive process.

Our general definition of consistency which follows does not require that development relations support any specific properties and we have considered the consequences of such unconstrained development elsewhere [2]. However, this paper is particularly concerned with strategies for incremental consistency checking and in order to obtain a rich enough theory to work with we will have to put some immediate constraints on development. Firstly, we assume all our development relations are reflexive. This is a natural requirement, although, it can be problematic for inter language consistency. We will say more about the position of inter language consistency later.

In addition to reflexivity, we will assume transitivity of development. This is slightly restrictive as it rules out implementation relations (e.g. LOTOS conf), but it is necessary in order to obtain a rich enough theory. In fact, the majority of the results we prove in the sequel require transitivity of development. In order to emphasize the importance of the transitivity assumption we include an example later. Example 1, which illustrates the damaging consequences of not making this assumption. As further justification, this paper is motivated by the search for incremental development strategies and transitivity of development seems a prerequisite of such incremental evolution of specifications. So, this paper assumes transitivity and reflexivity of the development relations used, i.e. they are preorders. So in fact, from this point on development and refinement are interchangeable terms.

We must also consider what interpretation of equivalence (which we denote ≃) we should adopt. A natural, and standard, interpretation is

\[ X \preceq_{dv} X' \text{ iff } X \vdash X' \land X' \vdash X \]

Thus, two descriptions are equivalent if and only if they are both developments of the other. \( \preceq_{dv} \) will play the role of identity in our theory. In the standard way, \( dv \) is a partial order with identity \( \preceq_{dv} \).

Another important property of equivalence is that two equivalent descriptions have identical development sets, i.e. every description that is a development of one will
be a development of the other. Furthermore, this situation only arises when the two descriptions are equivalent by $\preceq_{dv}$. This demonstrates that during system development we really can choose any one of a set of equivalent specifications without affecting the possibilities for future development.

In order to simplify presentation, we will consider strict development, i.e. relations $\overline{dv}$ which are subsets of the relations $dv$ with equivalence by $\preceq_{dv}$ factored out.

**Definition 1.** Overlining is an operation that can be applied to an arbitrary partial order, $dv$, with the following effect:

$$\overline{dv} = dv \setminus \preceq_{dv}.$$  

Note that $\overline{dv}$ is not reflexive, as all descriptions are equivalent to themselves.

Descriptions are written in formal techniques. A formal technique is characterised by the set of possible descriptions in the notation, a set of associated development relations and a set of semantic maps. For a particular formal technique $ft$ we denote the set of all descriptions in $ft$ as $DES_f$, the set of all development relations as $DEV_f$, and the set of all semantic maps as $SEM_f$.

**Basic definition.** In its general form consistency is a check which takes any number of descriptions, $X_1, X_2, \ldots, X_n$, and returns true if all the descriptions are consistent and false otherwise. This check will be performed according to a group of development relations, $dv_1, dv_2, \ldots, dv_n$, one per description, and is denoted

$$C(dv_1, X_1)(dv_2, X_2) \ldots (dv_n, X_n),$$

a shorthand for which is $C(dv_1, x_1)$. The validity of the check has two elements: type correctness and consistency.

Type correctness ensures that the consistency check being attempted is sensible. For example, it would prevent a development relation being applied to a specification written in a different language to that which the development relation is defined over. Type correctness becomes an issue when determining an appropriate inter language consistency check to apply. For simplicity, in this paper all consistency checks will be assumed to be type correct.

Intuitively we view $n$ specifications $X_1, X_2, \ldots, X_n$ as consistent if and only if there exists a physical implementation which is a realization of all the specifications, i.e. $X_1, X_2$ through to $X_n$ can be implemented in a single system. However, we can only work in the formal setting, so we express consistency in terms of a common (formal) description, $X$, and a list of development relations, $dv_1, dv_2, \ldots, dv_n$. **Definition 2** states that $n$ descriptions are consistent if and only if a description can be found which is a development of $X_1$ according to $dv_1$, $X_2$ according to $dv_2$, through to $X_n$ according to $dv_n$, and the description is internally valid, written $\Psi(X)$. The structure of the consistency check is depicted in Fig. 2 and is formalized in **Definition 2**. We denote this interpretation of consistency as $C$. 


Definition 2 (Consistency).

\[ C(d_{v_i}, X_i) \text{ holds, iff } \exists X \in DES \text{ s.t. } (X \land X_{\text{dv}_1} \land \cdots \land X_{\text{dv}_n} \land \Psi(X)). \]

The internal validity check in the above definition formalises the notion of implementability. It is required because descriptions relate to physical implementations in different ways for different languages and, in particular, for some FDTs not all specifications are implementable. For some FDTs it is possible to find a description which is a common development of a pair of specifications, but is not itself implementable. The property \( \Psi(X) \) is true if and only if the description \( X \) has a real implementation. Thus, \( \Psi \) acts as a receptacle for properties of particular languages that make descriptions in that language unimplementable. For example, a Z specification which contains contradictions would not be internally valid, e.g. an operation \([n! : n!] \land n! = 5 \land n! = 3 \) has no real implementation. This ensures that Definition 2 in the case that \( n = 1 \) coincides with what is commonly called “consistency” of a single specification. In fact, for an appropriate notion of development, \( \text{dv} \) say, we could write \( \Psi(X) \) as \( C(\text{dv}, X) \).

Unification is the mechanism by which descriptions are composed in such a way that the composition is a development of all the descriptions.

Definition 3 (Unification set).

\[ \mathcal{U}(d_{v_1}, X_1)(d_{v_2}, X_2) \cdots (d_{v_n}, X_n) = \{ X \in DES: X \land X_{\text{dv}_1} \land \cdots \land X_{\text{dv}_n} \land \Psi(X) \}. \]

We will use the notation \( \mathcal{U}(d_{v_1}, X_1) \) as a shorthand for \( \mathcal{U}(d_{v_1}, X_1)(d_{v_2}, X_2) \cdots (d_{v_n}, X_n) \). The unification set is the set of all common developments of a list of descriptions, i.e. the set of all unifications. Clearly, \( C(\text{dv}_i, X_i) \) holds if and only if \( \exists X \in \mathcal{U} \) such that \( \Psi(X) \). In fact, one approach to consistency checking is to perform a unification and then to show that this unification is internally valid.

The following proposition can be easily proved. The proposition expresses the obvious result that a unification of \( n \) specifications is a unification of a subset of the \( n \) specifications.
Proposition 1. \((\{dv_1, X_1\}, \ldots, (dv_n, X_n)\} \supseteq \{dv'_1, X'_1\}, \ldots, (dv'_m, X'_m)\) \Rightarrow \\
\forall (dv_1, X_1) \ldots (dv_n, X_n) \subseteq \forall (dv'_1, X'_1) \ldots (dv'_m, X'_m).

Our interpretation of consistency, \(C\), meets the requirements for a definition of consistency that we highlighted earlier, in the following ways:

- Different development relations can be instantiated which are appropriate both to different FDTs and to assessing different forms of consistency.
- Both intra and inter language consistency are incorporated. In particular, note that in most cases \(X_1, X_2, \ldots, X_n\) in the above definition will all be specifications, however, \(X\) will commonly be a semantic representation. In particular, if some of \(X_1, X_2, \ldots, X_n\) are in different languages then \(X\) is likely to be in a common semantic notation.
- Consistency checking between an arbitrary number of descriptions can be supported and checked according to a list of development relations. Binary consistency, e.g. \(C(dv_1, X_1)(dv_2, X_2)\), is just a special case of this global consistency. Binary consistency is a binary relation; we will often write it as \(X_1 C_{dv_1, dv_2} X_2\).
- Both balanced and unbalanced consistency are incorporated. Unbalanced consistency arises if \(dv_i \neq dv_j\) for some \(i \neq j\), while \(C(dv_i, X_i)\) is balanced if \(dv_i = dv_j\), \(\forall i, j\) s.t. \\
1 \leq i, j \leq n. Balanced consistency is written: \(C_{dv} X_i\) and binary balanced consistency, \(C_{dv}(X_1, X_2)\), is often written as \(X_1 C_{dv} X_2\).

It is beyond the scope of this paper to fully document the properties of our interpretation of consistency, the interested reader is referred to [2]; however, a number of classes of consistency will be used later in this paper and are reviewed now.

Complete consistency. It is possible that the application of a consistency check on a particular FDT may always be consistent, i.e. any set of descriptions chosen from the language will be consistent. This property is called complete consistency and is defined as follows.

**Definition 4 (Complete consistency of an FDT).** A formal description technique, \(ft\), is completely consistent according to a group of development relations \(dv_1, \ldots, dv_n\) if and only if for all \(X_1, \ldots, X_n\) in \(DES_{ft}\), \(C(dv_i, X_i)\).

Thus, if an FDT is known to be completely consistent there is no need to undertake consistency checking. This, for example, is the case for LOTOS specifications when balanced consistency according to extension or trace preorder are being considered. These examples will be returned to in Section 7.

Implementation complete. There are a number of languages in which all specifications are internally valid. This, for example, is the case with LOTOS\(^2\) and behavioural specification languages, such as LOTOS, Estelle and SDL, in general. We will discuss this aspect of the LOTOS language further in Section 3. Thus, we introduce the following notation.

\(^2\)There is an issue concerning inconsistencies within the LOTOS data language, i.e. inconsistent ACT-ONE algebraic rules. However, in this paper we will only be interested in the behavioural part of LOTOS. Thus, when we mention LOTOS in this paper, we are, to be more precise, talking about basic LOTOS.
Notation 1 (Implementation complete). A formal technique $f$ is called implementation complete iff $\forall X \in DES_f, \Psi(X)$.

Pairwise consistency. An important issue is in what way we can determine consistency, for example, can we assert consistency between three or more descriptions by performing a series of pairwise consistency checks. In order to determine this we consider the notion of a pairwise consistency check.

Definition 5 (Pairwise consistency). Descriptions $X_1, X_2, \ldots, X_n$ are pairwise consistent according to development relations $dv_1, dv_2, \ldots, dv_n$ iff

$$\forall i, j \text{ s.t. } 1 \leq i, j \leq n, X_i \subset dv_{i,j} X_j.$$ 

The following result characterizes the broad relationship between pairwise and normal consistency.

Proposition 2. (i) Consistency implies pairwise consistency.

(ii) Pairwise consistency of three or more specifications does not imply consistency.

Proof. (i) Assume $\exists X \in DES$ s.t. $(X \setminus dv_1 X_1 \land X \setminus dv_2 X_2 \land \cdots \land X \setminus dv_n X_n) \land \Psi(X)$. Now clearly $X_i \subset dv_{i,j} X_j$ for any $1 \leq i, j \leq n$ since $X$ can act as the internally valid common development.

(ii) We demonstrate this by counterexample. Consider the three specifications: $S_1 = [x!, y! : \mathcal{N} | x! = y!]$, $S_2 = [x!, z! : \mathcal{N} | x! = z!]$ and $S_3 = [z!, y! : \mathcal{N} | z! \neq y!]$. Intuitively these are balanced pairwise consistent, i.e. $S_1 \subset S_2$, $S_2 \subset S_3$, $S_1 \subset S_3$, but, they are not globally consistent. 0

Intuitively, the second part of the above proposition arises because pairwise consistency only requires the existence of a common development for each of the constituent binary checks. Thus, many binary consistency results may exist each of which focuses on a different common development. This is not sufficient to induce "global" consistency which requires the existence of a single common development.

3. Background on LOTOS

Subsequent sections of this paper apply the framework to the FDT LOTOS [3]. However, introducing LOTOS is beyond the scope of this paper, thus, this section will assume familiarity with the language. An introduction to LOTOS can be found in [3]. We reiterate the standard definitions of a number of the LOTOS development relations which we will use in this paper. First we introduce some notation.

Notation. In the following $P, P', Q, Q'$ stand for processes. $\mathcal{L}$ is the alphabet of observable actions associated with a certain process, while $i$ is the invisible or internal action. We use the variable $a$ to range over $\mathcal{L}$. Furthermore, $\mathcal{L}^*$ denotes strings (or traces) over $\mathcal{L}$. The constant $e \in \mathcal{L}^*$ denotes the empty string, and the variable $\sigma$ ranges over $\mathcal{L}^*$. In Table 1 the notion of transition is generalised to traces.
Using the notation derived in Table 1, we can define the following:

- \( T_r(P) = \{ \sigma \in \mathcal{L}^* \mid P \xrightarrow{\sigma} \} \), denotes the set of traces of a process \( P \).
- \( P \text{ after } \sigma = \{ P' \mid P \xrightarrow{\sigma} P' \} \), denotes the set of all states reachable from \( P \) by the trace \( \sigma \).
- \( \text{Ref}(P, \sigma) = \{ X \mid \exists P' \in (P \text{ after } \sigma), \text{ s.t. } \forall x \in X : P' \xrightarrow{\sigma} x \} \), denotes the refusals of \( P \) after \( \sigma \).

Also, we say that \( P \) has finite behaviour iff \( \exists n \in \mathbb{N} \forall \sigma \in Tr(P) : |\sigma| \leq n \) (where \( |\sigma| \) denotes the length of \( \sigma \)), otherwise we say it has infinite behaviour. This same concept appears in [22].

**Trace preorder.** An important category of system properties that one would like satisfied, are safety properties [10]. Safety properties state that something bad should not happen, where something bad can be interpreted as a certain trace of the specification. Observe that if \( S \) is a safety property, then \( \forall \sigma_1, \sigma_2 \) we have if \( \sigma_1 \leq \sigma_2 \) then \( S(\sigma_2) \Rightarrow S(\sigma_1) \), i.e. if \( S \) holds for the trace \( \sigma_2 \), it also holds for all its prefixes. In particular, all safety properties hold for the empty trace \( \varepsilon \).

When a specification is refined, it seems reasonable to require that the refinement is at least as safe as the specification. This intuition is reflected by the trace preorder.

**Definition 6 (Trace preorder).** Given two process specifications \( P \) and \( Q \), then \( P \) is a trace refinement of \( Q \), denoted \( P \xrightarrow{\sigma} Q \), iff \( Tr(P) \subseteq Tr(Q) \).

**Reduction.** In addition to safety properties the liveness (or deadlock) properties [10] of a system are also important. A liveness property states that something good must eventually happen. It may be required that a development of a specification does not deadlock in a situation where the specification would not deadlock, in other words, every trace that the specification must do, the development must do as well. A refinement relation that combines both the preservation of safety and liveness properties is the reduction relation, \( \text{red} \), defined in [10].

---

\(^3\) You should notice that this is just one interpretation/kind of refinement. The perspective that refinement corresponds to reduction of non-determinism is a common one, e.g. to take a non-process algebra example, strengthening postconditions corresponds to reducing non-determinism of state based specifications [6]. However, it is clearly only one interpretation and other valid notions of refinement exist, e.g. extension, which is discussed next.
Definition 7 (Reduction). Given two process specifications \( P \) and \( Q \), then \( P \) (deterministically) reduces \( Q \), denoted \( P \red Q \), iff:
1. \( P \leq_{tr} Q \), and
2. \( \forall \sigma \in Tr(Q), \Ref(P, \sigma) \subseteq \Ref(Q, \sigma) \).

Extension. Another refinement relation proposed in [10] is the extension relation. This relation allows new possible traces to be introduced, while preserving the liveness properties of the specification. Extension seems particularly relevant in the context of partial specification.

Definition 8 (Extension). Given two process specifications \( P \) and \( Q \), then \( P \) extends \( Q \), denoted \( P \ext Q \), iff:
1. \( Tr(P) \supseteq Tr(Q) \), and
2. \( \forall \sigma \in Tr(Q), \Ref(P, \sigma) \subseteq \Ref(Q, \sigma) \).

Conformance and its transitive closure. The \( \Conf \) relation [11], has been adopted as an interpretation of conformance in LOTOS, it is defined as follows.

Definition 9 (Conformance). \( P \conf Q \), iff \( \forall \sigma \in Tr(Q), \Ref(P, \sigma) \subseteq \Ref(Q, \sigma) \).

However, as suggested previously \( \Conf \) is unfortunately not transitive [24, 15]. Thus, we will use the transitive closure of \( \Conf \), which we denote \( \Conf^* \).\(^5\) Our use of this relation is justified by recent work [15] which suggests that \( \Conf^* \) is an important relation in its own right which can be characterised in terms of a notion of testing called friendly testing. We will use \( \Conf^* \) for illustrative purposes later in the paper. When we do this we will need to use some simple properties of \( \Conf^* \) which we prove here.

Proposition 3. (1) \( \Conf \subseteq \Conf^* \);  
(2) \( \Red \subseteq \Conf^* \cap \leq_{tr} \).

Proof. The first of these follows from the fact that \( \Conf \) is not transitive, see for example [24]. The second needs some explanation.

Firstly, we can see that \( \Red \subseteq \Conf^* \cap \leq_{tr} \) since from (1), \( \Conf \cap \leq_{tr} \subseteq \Conf^* \cap \leq_{tr} \).

Secondly, we can provide an example which justifies that \( \Conf^* \cap \leq_{tr} \neq \Red \). Consider the following three processes (these are the processes used by [24] to show that \( \Conf \) is not transitive):

\[
R_1 := b;\ stop \ [\ i; a;\ stop \\
R_2 := i; a;\ stop \\
R_3 := b; c;\ stop \ [\ i; a;\ stop
\]

\(^4\) Note that the definition of refusal ensures that if \( \sigma \not\in Tr(P) \) then \( \Ref(P, \sigma) = \emptyset \).

\(^5\) Note that \( \Conf \) is reflexive.
It can be shown that $R_1 \text{conf} R_2$, $R_2 \text{conf} R_3$ and $-(R_1 \text{conf} R_3)$. Thus, $R_1 \text{conf}^* R_3$ and furthermore, $R_1 \text{conf}^* \cap \preceq_{VR} R_3$ but $-(R_1 \text{red} R_3)$. □

*Testing equivalence.* A standard interpretation of equivalence is given by the testing equivalence relation [10]. As its defining properties we could use either that $P \text{ te } Q \Leftrightarrow P \text{ red } \cap \text{ red}^{-1} Q \Leftrightarrow P \text{ ext } \cap \text{ ext}^{-1} Q$. So, testing equivalence is a sensible identity in the sense of $\approx$ for both red and ext.

**Definition 10** (Testing equivalence).

$$P \text{ te } Q \text{ iff } Tr(P) = Tr(Q) \land \forall \sigma \in Tr(P), \text{ Ref}(P, \sigma) = \text{ Ref}(Q, \sigma).$$

Note that all the relations we have defined here are preorders (apart from conf). In addition as its name suggests, testing equivalence is also symmetric.

*Internal validity.* At least theoretically, we can view all LOTOS specifications as implementable (if we once again ignore the possibility of inconsistent data definitions). Even degenerate specifications, such as those containing deadlocks, for example, have a physical implementation equivalent. Thus, all LOTOS descriptions are internally valid. This is a fundamental characteristic of behavioural languages that distinguishes them from logically based specification notations.

**Proposition 4.** (Assuming consistency of the ACT-ONE definitions) LOTOS is implementation complete.

This proposition is important as it considerably simplifies the class of consistency that must be considered for LOTOS.

4. Basic strategies for consistency checking

Up to this point we have investigated consistency in terms of a set of possible unifications, i.e. descriptions $X_1, X_2, \ldots, X_n$ are consistent if, firstly, the set of possible unifications $\mathcal{U}^*(d_{v_i}, X_i)$ is non-empty and, secondly, the set contains an internally valid description. Such a unification set could be very large and often infinite. Clearly, if a system development trajectory is to be provided for viewpoint models then it is important that we reduce the choice of unification. Ultimately, we would like to select just one description from the set of unifications. This will not always be possible. So, we will also investigate how we can work with a subset of the unification set. This will enable us to obtain global consistency from a series of non-global (probably binary) consistency checks and unifications. The objective of the remainder of this paper is to characterise the unification(s) that should be chosen from the unification set.

This section considers basic strategies for consistency checking. In particular, the issue of representative unification is considered in Section 4.1. Then general formats for binary consistency checking are considered in Section 4.2 and the central issue of
minimally developed unification is discussed in Section 4.3. These basic strategies will be used in later sections when we consider the properties required in order to realise a binary consistency checking strategy.

4.1. Representative unification

A particular unification algorithm will construct just one member of the unification set. Importantly, we need to know that the unification that we construct is internally valid if and only if an internally valid unification exists; otherwise we may construct an internally invalid unification despite the fact that an alternative unification may be internally valid.

Thus, we introduce the concept of a representative unification, which is defined as follows.

Definition 11. $X \in \mathcal{U} (dv, X_i)$ is a representative unification iff

$$\exists X' \in \mathcal{U} (dv, X_i) \text{ s.t. } \Psi(X') \Rightarrow \Psi(X)$$

The following result is very straightforward.

Proposition 5. $ft$ is implementation complete and $X_1, \ldots, X_n \in DES_f \Rightarrow$

$$\forall X \in \mathcal{U} (dv, X_i) X \text{ is a representative unification.}$$

So, this result implies that for a language such as LOTOS, representativeness of unification does not arise.

We would certainly expect the unification functions that we adopt to yield a representative unification as a reflection of this, for the remainder of this paper we will assume representativeness of the unification functions that we consider.

4.2. Binary consistency checking strategies

We would like to obtain global consistency through a series of binary consistency checks. We have found that naive pairwise checking does not give us this, see Proposition 2. However, a combination of binary consistency checks and binary unification of the form shown in Fig. 3 should intuitively work, i.e. $X_1$ and $X_2$ are checked for consistency, then a unification of $X_1$ and $X_2$ is obtained, which is checked for consistency against $X_3$, then a unification of $X_3$ and the previous unification is performed. This process is continued through the $n$ viewpoint descriptions. Thus, the base case is a binary consistency check and then repeated unification and binary consistency checks are performed against the next description. Of course, this is just one possible sequence of binary consistency checks. We would like to obtain full associativity results which support any appropriate incremental consistency checking strategy. However, as an archetypal approach, the binary consistency checking strategy of Fig. 3 will serve as an initial focus for our investigations.
The advantages of such incremental consistency checking strategies are that they do not force the involvement of all viewpoints in every consistency check. In particular, it may be possible to incrementally correct inconsistencies. In addition, such an approach will aid maintaining structure when unifying. One of the main problems with unification algorithms is that the generated unification will, in general, be devoid of high level specification structure (e.g. operators such as $[]$ in LOTOS are expanded out) [29]. This is a big problem if the unification is to be further developed. It is very unlikely that a single unification of a large group of viewpoints will be able to reconcile the structure of all the views, however, an incremental focus of restructuring may be possible.

The next definition characterises the binary consistency checking strategy that we are interested in. We denote the strategy $I_U$, where $U$ is a particular binary unification function. $I$ incorporates a series of binary consistency checks, each of which uses $U$ to perform the binary unification. $U$ has the general form

$$U : (DEV \times DES) \times (DEV \times DES) \rightarrow \mathbb{P}(DES)$$

i.e. it takes two pairs (each comprising a development relation and a description) and yields a set of descriptions. A typical application of the function, e.g.

$$U(dv,X)(dv',X')$$

generates a set of descriptions, which are, intuitively, possible unifications of $X$ and $X'$ according to $dv$ and $dv'$ respectively. We will investigate the suitability of specific binary unification functions by instantiating these functions for $U$ in $I_U$. Thus, $I_U$ gives us a general structure for obtaining global consistency from a series of binary unifications, but it is parameterised on the particular binary unification function to use. Obviously, the spectrum of possible instantiations of $U$ is very large, from functions that yield all possible binary unifications, i.e. $\mathcal{U}$, to functions which select just one
unification. Clearly, our ultimate objective is to use a unification function which yields a single unification, however, this will not turn out to be possible in all cases. Thus, at this stage we have chosen to be most general and let $U$ generate a set of unifications.

**Definition 12.**

$$
\Pi_U(dv_1, X_1) \ldots (dv_n, X_n) \overset{def}{=} \\
(\exists Y_1 \in U(dv_1, X_1)(dv_2, X_2) \land \Psi(Y_1)) \land \\
(\exists Y_2 \in U(dv_1 \cap dv_2, Y_1)(dv_3, X_3) \land \Psi(Y_2)) \land \\
(\exists Y_3 \in U(dv_1 \cap dv_2 \cap dv_3, Y_2)(dv_4, X_4) \land \Psi(Y_3)) \land \\
\ldots \\
(\exists Y_{n-2} \in U(dv_1 \cap dv_2 \cap \ldots \cap dv_{n-2}, Y_{n-3})(dv_{n-1}, X_{n-1}) \\
\land \Psi(Y_{n-2})) \land \\
(\exists Y_{n-1} \in U(dv_1 \cap dv_2 \cap \ldots \cap dv_{n-1}, Y_{n-2})(dv_n, X_n) \\
\land \Psi(Y_{n-1})))
$$

Step 1

Step 2

Step 3

Step $n-2$

Step $n-1$

Thus, each step in the algorithm considers a unification set using the binary unification function $U$. The $i$th step is satisfied if a description, $Y_i$, can be found in the set of unifications generated by the function $U$ that is internally valid and can be used to satisfy the $(i+1)$th step. A depiction of $\Pi_U$, with $n = 4$, is given in Fig. 4. It should be apparent that consistency checking is implicit in each step. Thus, the existence of an internally valid $i$th unification, $Y_i$, ensures that $Y_{i-1}$ and $X_{i+1}$ are consistent. Clearly, if an internally valid unification does not exist for a particular step then consistency would be lost.

In addition notice how when we have performed a binary unification, e.g. $U(dv, Y)$ ($dv, X_i$), the next binary unification will intersect $dv$ and $dv_i$; this ensures that the final unification (using transitivity of development) is a development by $dv_j$ of $X_j$ for all $j$.

As mentioned earlier the unification construction function, $U$, yields a set of unifications, which could be a singleton. We assume $U$ satisfies the following constraints.

**Definition 13.** A binary unification function $U$ is valid if and only if

$$(U.i) \ U(dv, X)(dv', X') \subseteq \forall(dv, X)(dv', X') \text{ and}$$

$$(U.ii) \ \forall(dv, X)(dv', X') \neq \emptyset \Rightarrow U(dv, X)(dv', X') \neq \emptyset.$$

These are minimal constraints that ensure $U$ is a sensible binary unification method. ($U.i$) guarantees that the unifications generated by $U$ are in the set of all unifications obtained by $\forall$ (remember $\forall$ is our base unification function, see Definition 3) and ($U.ii$) ensures that if a unification exists, $U$ will not yield the empty set. Using these constraints we can show that if our binary consistency checking strategy is satisfied then consistency follows.
Proposition 6. Assuming $d_{vi}$, $1 \leq i \leq n$, is a preorder and $U$ satisfies (U.i) and (U.ii),

$$\Pi_U(d_{vi}, X_i) \Rightarrow C(d_{vi}, X_i).$$

Proof. Assume $\Pi_U(d_{vi}, X_i)$ holds. Now from step $n - 1$ in $\Pi_U$ we deduce

$$\exists Y \ s.t. \ Y (d_{v1} \cap d_{v2} \cap \cdots \cap d_{vn-1}) Y_{n-2} \land$$

$$Y \ d_{vn} X_n \land$$

$$\Psi(Y)$$

We will show that $Y$ is the required common development of $X_1$ through to $X_n$ to give us $C(d_{vi}, X_i)$. Firstly, (2) and (3) give us immediately that $\Psi(Y)$ and $Y \ d_{vi} X_n$. Now from (1) and $Y_{n-2} \in U(d_{v1} \cap \cdots \cap d_{vn-2}, Y_{n-3})(d_{vn-1}, Y_{n-1})$ we can deduce that $Y \ d_{vn-1} Y_{n-2}$ and $Y_{n-2} \ d_{vn-1} X_{n-1}$, thus, from transitivity of $d_{vn-1}$ we have $Y \ d_{vn-1} X_{n-1}$. We can perform similar arguments down through the construction of $\Pi$ to determine that $Y \ d_{vn-2} X_{n-2} \land \cdots \land Y \ d_{v2} X_2 \land Y \ d_{v1} X_1$. Thus, $Y$ is the required common development and $C(d_{vi}, X_i)$ holds. \qed

Example 1. It is here that we can really illustrate why transitivity of development relations is so essential, we do this by giving an example of an invalid consistency
checking strategy that could arise with the, non-transitive, development relation conf. So, consider the following LOTOS specifications:

\[ P_1 := b; \text{stop} \left[ i \right] i; \text{a stop} \]
\[ P_2 := \text{stop} \]
\[ P_3 := b; c; \text{stop} \left[ i \right] i; \text{a stop} \]
\[ P_4 := i; a; \text{stop} \]

The following relationships under conf are key to the example:

\[ \forall j(1 \leq j \leq 4). P_j \text{ conf } P_2 \]
\[ P_4 \text{ conf } P_3, P_1 \text{ conf } P_4 \]

but, importantly, \( -(P_1 \text{ conf } P_3) \) because after the trace b, \( P_1 \) refuses c while \( P_3 \) does not refuse it. In fact, \( P_1, P_4 \) and \( P_3 \) are the canonical examples of non-transitivity of conf once again.

Now consider the following consistency check:

\[ C(\text{te}, P_1)(\text{conf}, P_2)(\text{conf}, P_3) \]

It is clear that this cannot hold since any process that is testing equivalent to \( P_1 \) cannot conform to \( P_3 \) (as just highlighted). However, due to non-transitivity of conf applying a consistency checking strategy in the style of \( II \) will incorrectly find the three specifications consistent. Such a strategy is shown in Fig. 5.

It is also worth pointing out that associative unification strategies for checking balanced consistency under te and conf can easily be defined, see Proposition A.1 in the appendix. However, this fact does not ensure that when we put the two relations together in an unbalanced consistency check, we will get a well behaved consistency checking strategy.

Using Proposition 6 we can show that performing \( II \) with the full unification set function, i.e. instantiating \( \mathcal{U} \) for \( U \), is equal to consistency. Clearly, we would expect this to be the case and if it was not we would have to worry about \( II \).
Proposition 7. Assume $d_{v_i}$, $1 \leq i \leq n$, is a preorder. Then,

$$
\prod_{i=1}^{n} (d_{v_i}, X_i) = C(d_{v_i}, X_i).
$$

Proof. ($\Rightarrow$) $\mathcal{U}$ trivially satisfies (U.i) and (U.ii); thus we can use the previous result, Proposition 6, to give this direction of implication.

($\Leftarrow$) Assume $C(d_{v_i}, X_i)$ holds, i.e. $\exists Y \in \mathcal{U}(d_{v_i}, X_i)$ such that $\Psi(Y)$. We will show that $Y$ can act as the unification in all steps of $\Pi$. Firstly, the internal validity requirement of each step will clearly be satisfied for $Y$. In addition, using Proposition 1 we get, $Y \in \mathcal{U}(d_{v_1}, d_{v_2}, Y)(d_{v_3}, Y)$ and thus step 1. Step 2 follows since $Y$, by our assumption and $Y \in \mathcal{U}(d_{v_1}, d_{v_2}, Y)$, $d_{v_3}$, by our assumption and $Y \in \mathcal{U}(d_{v_1}, d_{v_2}, Y)$. Using similar arguments we can get step 3 and all steps up to $\Pi - 1$ as required. 

However, if we use a valid unification construction function (i.e. one that satisfies (U.i) and (U.ii)) other than $\mathcal{U}$ the converse to Proposition 6 does not, in general, follow, i.e. $C \not\Rightarrow \Pi_U$, and we clearly require this direction if $\Pi$ is to be used.

Example 2. We will give two straightforward examples of why a simple binary consistency checking strategy may not give global consistency. The first example is for LOTOS and the second is for Z.

LOTOS. Consider the three LOTOS specifications, $P_1 := i; a; \text{stop} [] i; b; \text{stop}$, $P_2 := a; \text{stop} [] i; b; \text{stop}$ and $P_3 := a; \text{stop}$. Further consider the consistency check $\mathcal{C}_{\text{red}}(P_1, P_2, P_3)$, where $\mathcal{C}_{\text{red}}$ is the LOTOS reduction relation, which refines through reduction of non-determinism (see Section 3). The three specifications are consistent by reduction since $P_3$ is a reduction of all three specifications. However, if we attempt a binary consistency checking algorithm and started with $P_1$ and $P_2$ we may choose as the unification of these two the process $P := i; b; \text{stop}$, and $\mathcal{C}_{\text{red}}(P, P_3)$ does not hold.

Z. Consider the three Z specifications, $S_1 = [n! : \text{false} \mid n! = 5 \lor n! = 7]$, $S_2 = [n! : \text{false} \mid n! = 7 \lor n! = 5]$, and $S_3 = [n! : \text{false} \mid n! = 5]$. The first two specifications could be unified to yield $[n! : \text{false} \mid n! = 7]$, which is not consistent with the third. But, the third specification could act as a refinement of all three.

These examples suggest the class of unifications that we must select. Specifically, we should choose the least developed unification, i.e. the one that is most abstract and is, in terms of development, closest to the original descriptions. In both the above examples this will give the required result. In the LOTOS example $P_2$ itself should have been chosen as the unification of $P_1$ and $P_2$ as it is the least reduced unification, up to testing equivalence. Similarly, in the Z example either of the identical specifications $S_1$ or $S_2$ should have been chosen initially. The issue is that we could choose a unification of two descriptions that is too developed to be reconciled with a third description, while a less developed unification that could be reconciled, exists. The problem is evolving the two original specifications unnecessarily far towards the concrete during unification.
We will in fact reserve the term least developed unification for a very well behaved class of consistency, we have a few more hurdles before we get to it. Thus, the next section considers the more general concept of a minimally developed unification.

4.3. Minimally developed unifications

In traditional single threaded (waterfall) models of system development the issue of minimal/least development does not arise. This is because, assuming development is a preorder, each description is a least development of itself, i.e. is a development of itself (because of reflexivity) and is less developed than any other development. Unfortunately, the situation is not so straightforward when we generalise to viewpoints and when we must reconcile the development trajectory of more than one description.

First our interpretation of minimally developed unification. We assume \( d_{v_1}, \ldots, d_{v_n} \) are preorders.

**Definition 14 (Minimally developed unification).** \( X \in \mathcal{U}(d_{v_1}) \) is a minimally developed unification iff \( \exists X' \in \mathcal{U}(d_{v_1}) \) s.t. \( X \cap d_{v_1} X' \), where \( \cap d_{v_1} \) is a shorthand for \( d_{v_1} \cap \cdots \cap d_{v_n} \).

This definition ensures that a unification which \( X \) is a strict development of does not exist. Notice the interpretation of development, that \( X \) and \( X' \) are related by \( d_{v_1} \cap \cdots \cap d_{v_n} \), i.e. the set of unifications is ordered by the intersection of the development relations used in unification. Fig. 6 depicts a typical situation, \( X, X' \) and \( X'' \) are unifications of \( X_1 \) and \( X_2 \) and \( X, X' \) and \( X'' \) are ordered by \( d_{v_1} \cap d_{v_2} \). In this diagram \( X \) is the minimally developed unification of \( X_1 \) and \( X_2 \). \( d_{v_1} \cap \cdots \cap d_{v_n} \) is a natural interpretation of development between unifications because all descriptions in

![Fig. 6. A typical minimally developed unification situation.](image-url)
the unification set that are descendents of a minimally developed unification $X$ are developments of $X$ by all relevant development relations.

Note that another way of looking at the minimally development is that it is a maximal element in the set of possible developments. Thus, by reversing the point of reference we can exchange minimally for maximal. At some points in the text it will be most convenient to make this reversal and talk in terms of maximal elements of sets of developments.

Unfortunately, for inter language consistency, the minimally developed of the set of unifications is a problematic concept. Specifically, descriptions in the unification set, $U(dv_1, X_i)$, are likely to be in a different notation from $X_1, \ldots, X_n$; thus it is unlikely that the unifications can be related in a type correct manner using $dv_1 \cap \cdots \cap dv_n$. Thus, this definition and the remaining theory will only be applied to intra language consistency. Ongoing work is addressing generalisation of minimally developed unification to the inter language setting.

It is also disappointing to discover that for arbitrary development relations (even when constrained to be preorders and in the intra language setting) the minimally developed unification will not necessarily be unique. (By way of clarification, here we are talking about uniqueness up to equivalence, where equivalence is interpreted as $\equiv_{dv_1, \ldots, dv_n}$ for $dv_1, \ldots, dv_n$ the relevant development relations. Throughout the remainder of the paper, when we talk about uniqueness, we mean unique up to equivalence.)

**Example 3.** If we have four descriptions: $X_1$, $X_2$, $X_3$ and $X_4$; and the development relations between descriptions indicated in Fig. 7, both $X_3$ and $X_4$ are minimally developed unifications of $X_1$ and $X_2$, i.e. they are clearly both in $U(dv_1, X_1)(dv_2, X_2)$ and neither has an ancestor by $dv_1 \cap dv_2$ in $U(dv_1, X_1)(dv_2, X_2)$. Furthermore, examples of this form are characteristic of situations that foil $\Pi$. Specifically, consider the development relations in Fig. 8. In this situation we may unify $X_1$ and $X_2$ to $X_4$ and then

---

**Fig. 7. Development relations.**
Fig. 8. Problematic unifications.

Fig. 9. LOTOS development relations.

fail to find a common development with $X_3$ even though $X_5$ could act as the required common development of $X_1$, $X_2$, and $X_3$.

We can illustrate such a situation by considering the LOTOS consistency check $C(\leq_{tr}, X_1)(\text{ext}, X_2)(\leq_{tr}, X_3)$, where

$$X_1 := a; b; c; \text{stop} \quad X_2 := a; \text{stop} \quad X_3 := a; \text{stop} \quad X_4 := a; b; \text{stop}$$

Now both $X_2$ and $X_4$ (amongst others) are minimal unifications from $\mathcal{U}(\leq_{tr}, X_1)$ (ext, $X_2$) (Fig. 9 shows a subset of the relevant development relations). In particular, notice that $X_2$ and $X_4$ are not related by $\leq_{tr} \cap \text{ext}$, thus, neither is less developed than the other. Furthermore, if we now consider the full consistency check, $C(\leq_{tr}, X_1)(\text{ext}, X_2)(\leq_{tr}, X_3)$, the choice of minimally developed unification is extremely significant, since $X_2 \leq_{tr} X_3$ but $X_4 \not\leq_{tr} X_3$. Thus, in order to perform this consistency check we need to check against the set of all minimally developed unifications of $X_1$ and $X_2$, which would include both $X_2$ and $X_4$.

In response to these observations we will divide our discussion of minimally developed unification into two parts. First, we will consider the situation in which the minimally developed unification is not unique, then we will discuss the situation in which it is unique. These two cases will be discussed in the following two sections. In the former case we consider unification according to the set of all minimally developed unifications. This is a compromise of our ultimate objective which is to locate a single unification, but it allows us to, in general, reduce the specification set to some extent.
Our objective is to consider the consequence of using the minimally developed unification set as unification function. If this gives us the required relationship between Π and C, then we will attempt to be more selective from amongst the minimally developed unification set and locate under what circumstances we can take just one element from the set.

5. Non-unique minimally developed unification

5.1. Relating consistency and minimal development

We define the minimally developed unification set, which we denote $\mathcal{M}\mathcal{U}(dv_1,X_1)\ldots (dv_n,X_n)$ or $\mathcal{M}\mathcal{U}(dv,X)$, as follows.

Definition 15 (Minimally developed unification set).

$$\mathcal{M}\mathcal{U}(dv_1,X_1) = \{X: X \in \mathcal{U}(dv_1,X_1) \land \neg(\exists X' \in \mathcal{U}(dv_1,X_1), \text{ s.t. } X \cap dv_1 X')\}.$$ 

Thus, the minimally developed unification set is the set of all unifications that do not have a non-equivalent ancestor in the unification set. In order to use $\mathcal{M}\mathcal{U}$ as the unification function in Π we must show that $\mathcal{M}\mathcal{U}$ is valid with regard to $\mathcal{U}$, i.e. it satisfies conditions (U.i) and (U.ii). The first of these is straightforward, it follows directly from the next proposition.

Proposition 8. $\mathcal{M}\mathcal{U}(dv_1,X_1) \subseteq \mathcal{U}(dv_1,X_1)$.

Proof. Take $X \in \mathcal{M}\mathcal{U}(dv_1,X_1)$, by the definition of $\mathcal{M}\mathcal{U}$, $X \in \mathcal{U}(dv_1,X_1)$. □

Corollary 1. $\mathcal{M}\mathcal{U}(dv,X)(dv',X') \subseteq \mathcal{U}(dv,X)(dv',X')$.

(U.ii) though is more difficult and obtaining this validity constraint is central to showing that $\Pi_{\mathcal{M}\mathcal{U}}$ is equal to C. We will have to impose certain “well behavedness” constraints on development in order to obtain this property. With the constraints that we have already imposed on development, i.e. being a preorder, these properties give us a set of requirements that development in a particular formalism must satisfy in order for it to be used in our framework of unification. In order not to lose the flow of our current argument we will refrain for the moment from consideration of these constraints; they will be discussed in Section 5.2. For the moment we simply state the result that we want; Section 5.2 will provide proofs. We actually need a stronger property than (U.ii) in order to prove the forthcoming Theorem 1. The property that we need is Property 1.
Property 1. \( X \in \mathcal{U}(dv_i, X_i) \Rightarrow \exists X' \in \mathcal{U}(dv_i, X_i) \text{ s.t. } X \cap dv_i X' \).

This property states that all unifications have an ancestor in the minimally developed unification set. In other words, all unifications are developments, by \( dv_i \), of a minimally developed unification. Notice, a minimally developed unification is a development of itself. You may think that such a requirement would naturally hold, but Section 5.2 shows that this is not the case. Once we have Property 1 we can easily obtain (U.ii); it is an immediate consequence of the following, which can be easily verified.

Proposition 9. Property 1 \( \Rightarrow (\mathcal{U}(dv_i, X_i) \neq \emptyset \Rightarrow \mathcal{U}(dv_i, X_i) \neq \emptyset) \).

We now have enough theory to tackle the main concern of this section; obtaining global consistency from binary consistency checking and to relate \( C \) to \( H_{\#} \).

Theorem 1. Given Property 1 and \( \mathcal{U} \) a representative unification.

\[
C(dv_i, X_i) = \Pi_{\#}(dv_1, X_1) \ldots (dv_n, X_n).
\]

Proof. The first section of the appendix contains an induction proof (Proposition A.2), where \( \Pi_{\#} \) is a slightly stronger constraint than \( \Pi_{\#} \), that \( C(dv_i, X_i) \rightarrow \Pi_{\#}(dv_1, X_1) \ldots (dv_n, X_n) \). From an examination of the conditions of \( \Pi_{\#} \), if \( \mathcal{U} \) is representative, \( \Pi_{\#}(dv_1, X_1) \ldots (dv_n, X_n) \rightarrow \Pi_{\#}(dv_1, X_1) \ldots (dv_n, X_n) \). In addition, Proposition 6 gives us the other direction of implication. \( \square \)

This is the result we are seeking, it states that subject to Property 1 holding and \( \mathcal{U} \) a representative unification strategy we can equate the binary consistency checking strategy \( \Pi_{\#} \) with consistency, i.e. if each binary unification considers the set of all minimally developed unifications then we will obtain consistency. However, in order, to obtain this identity it is sufficient to verify Property 1. The next subsection considers constraints on development that realise this property.

5.2. Constraints on development

The difficulty surrounding obtaining Property 1 (and hence constraint (U.ii)) is that the chain of candidate minimal unifications may be infinite, as depicted in Fig. 10 and a maximal member of the chain, \( Y_i \), may not exist. We can illustrate this situation using the LOTOS development relations, we do this in the following example.

Example 4. Let

\( \leq = \text{conf}^* \cap \sim \)

where \( \sim \) is the equivalence formed as follows:

Let \( K \subset \text{DES}_{\text{LOTOS}} \) be the set of all LOTOS specifications that have finite behaviour, e.g. \( a; a; \text{stop} \) and \( b; \text{stop} []; c; a; \text{stop} \), would both be in \( K \), but \( P := a; P \) would not be in \( K \) and let \( \sim = K \times K \).
It is straightforward to show that $\leq$ is a preorder.

The intuition behind $\leq$ is that $P_1 \leq P_2$ means that $P_1$ conforms (+ a bit more in order to get transitivity) to $P_2$ and, in addition, $P_1$ cannot be infinite (in terms of traces) unless $P_2$ is. This is a form of realizability constraint which ensures that $\text{conf}^*$ does not hinder realizability, i.e. if $P_2$ is finite and thus, in a very strong sense, realizable, it ensures that $P_1$ will also be finite and similarly realizable.

Now let

$$P := a; P \quad \text{and} \quad Q := \text{stop}$$

then the following consistency check:

$$C(P, \leq_r)(Q, \leq)$$

does not yield a well formed minimal unification set. For example, let $Y$ be the infinite chain of processes, informally, characterised as follows:

$$Y_0 := i; \text{stop}$$
$$Y_1 := i; \text{stop} \; a; \text{stop}$$
$$Y_2 := i; \text{stop} \; a; (i; \text{stop} \; a; \text{stop})$$
$$Y_3 := i; \text{stop} \; a; (i; \text{stop} \; a; (i; \text{stop} \; a; \text{stop}))$$

... .

It can be seen that $Y \subseteq \mathcal{H}(P, \leq_r)(Q, \leq)$, since if we take $R$ as an arbitrary element from $Y$ then,

1. $\text{Tr}(P)$ is the set of all finite prefixes of the infinite trace $a \; a \; a \; \ldots$ and clearly $\text{Tr}(R) \subseteq \text{Tr}(P)$,

2. $Q$ is finite and all elements of $Y$ are finite and also $Q$ has just one trace, $c$, and $\text{Ref}(Q, c) = \mathcal{P}(\text{Act})$, so, for all $R \in Y$. $\text{Ref}(R, c) \subseteq \text{Ref}(Q, c)$ and thus, $R \text{conf}^* Q$ (since $\text{conf} \subseteq \text{conf}^*$).

In addition, $\leq_r \cap \leq = (\leq_r \cap \text{conf}) \cap \circ (\leq_r \cap \text{conf}) \cap \sim = \text{red} \cap \sim$. So, for all finite processes $\leq_r \cap \leq \subseteq \circ \text{red}$. Furthermore, all elements of $Y$ are finite, $Y_i \text{ red} Y_{i+1}$.
(and hence $Y_i \leq_Y \cap \leq Y_{i+1}$) and $Y$ does not contain an upper bound according to $\leq_Y \cap \leq$. Thus, $Y$ is an infinitely increasing chain of candidate minimal unifications.

However interestingly, both $(DESLOTS, \leq)$ and $(DESLOTS, \leq_Y)$ satisfy Property 1 individually. A proof of the latter of these is straightforward. We present a proof of the former in the appendix: Proposition A.3.

The following proposition shows that if all infinite chains in the unification set have an upper bound, i.e.

$$\{X_1, X_2, \ldots\} \subseteq \mathcal{U}(dv_i, X_i)$$

is an infinite chain $\Rightarrow$

$$\exists X \in \mathcal{U}(dv_i, X_i), \forall j . X_j \cap dv_i X$$

then Property 1 will hold. Remember, Property 1 states

$$X \in \mathcal{U}(dv_i, X_i) \Rightarrow \exists X' \in \mathcal{U}(dv_i, X_i) s.t. X \cap dv_i X'$$

i.e. all unifications are descendents of a minimally developed unification.

Proposition 10. (i) All infinite chains in $\mathcal{U}(dv_i, X_i)$ have an upper bound $\Rightarrow$

(ii) $X \in \mathcal{U}(dv_i, X_i) \Rightarrow \exists X' \in \mathcal{U}(dv_i, X_i) s.t. X \cap dv_i X'$. i.e. (i) $\Rightarrow$ Property 1.

Proof. By contradiction, so, assume (i). Now $\neg$(ii) gives:

$$\exists X \in \mathcal{U}(dv_i, X_i) s.t. \neg(\exists X' \in \mathcal{U}(dv_i, X_i) s.t. X \cap dv_i X').$$

Now consider the following construction:

1. $Y_0 = X$.
2. Select $Y_1 \in \mathcal{U}(dv_i, X_i)$ such that $Y_0 \cap dv_i Y_1$. Such a $Y_1$ must exist, otherwise $Y_0$ would be a minimally developed unification and a development by $\cap dv_i$ of itself, which contradicts our assumption of $\neg$(ii).
3. If $Y_0, Y_1, \ldots, Y_j \in \mathcal{U}(dv_i, X_i)$ for $j \geq 0$, such that $Y_0 \cap dv_i Y_1 \cap dv_i Y_2 \cap \cdots \cap dv_i Y_{j-1}$ have already been chosen, then a description $Y_{j+1}$ such that $Y_j \cap dv_i Y_{j+1}$ can be found. Such a $Y_{j+1}$ must exist otherwise $Y_j$ would be a minimally developed unification and by transitivity of development an ancestor by $\cap dv_i$ of $Y_0$, which would contradict our assumption of $\neg$(ii).

This construction will generate an infinite ascending chain by $\cap dv_i$ of descriptions $Y_0, Y_1, Y_2, \ldots \in \mathcal{U}(dv_i, X_i)$. Property (i) implies that a $W$ must exist such that $\forall Y_i, Y_j$.

---

In previous work [4], we have assumed a stronger property than this, that of well foundedness. However, working with upper bounds simplifies the theory and turns out to be a more useful constraint.
$\cap \cup v_i \ W$. Now this $W$ may be in an infinite chain which will in turn have an upper bound. Furthermore, this series of infinite chains must terminate at an upper bound, $Z_0$ say, that is not in an infinite chain, otherwise (i) would be contradicted. Now $Z_0$ can only have a finite chain of specifications $Z_j$ such that $Z_j \cap \cup v_i \ Z_{j+1}$, i.e. there must exist an $n$ such that $\exists Z$ such that $Z_n \cap \cup v_i \ Z$. Thus, $Z_n \in \cup \cup (\cup v_i, X_i)$ and by transitivity $X \cap \cup v_i \ Z_n$, which contradicts our assumption of $\neg$(ii). $\square$

Using this result we can obtain the following important corollary.

**Corollary 2.** (i) All infinite chains in $(\cup (\cup v_i, X_i), \cup \cup v_i)$ have an upper bound.

$\Rightarrow$

(ii) Property 1.

This result characterises the properties that are required of $\cup (\cup v_i, X_i)$ in order to obtain Property 1. In order to use a particular FDT we would actually like to know that any combination of development relations and descriptions in the language will yield a unification set that satisfies Property 1. We will clearly obtain this if an FDT upholds the following property.

**Property 2.** FDT $ft$ satisfies Property 2 iff

$$\forall X_j \in DES_{ft}, \ dv_k \in DEV_{ft}, \ all \ infinite \ chains \ in$$

$$(\cup (\cup v_i, X_i), \cup \cup v_i) \ have \ an \ upper \ bound.$$ 

So, by way of summary, by imposing this property we ensure that minimally developed unifications exist. In other words, if we flip our point of reference, the unification set has maximal elements. Hence, we can check consistency by taking the set of minimally developed unifications at each stage.

### 6. Unique minimally developed unification

Clearly, we would like to unify to a single description. So far, we have only considered situations in which we have to test every element of a set of unifications in order to obtain global consistency. Although, the set of minimally developed unifications is likely to be significantly smaller than the full unification set, it could still be very large. In fact, it could still be infinite if the set of minimally developed unifications contains an infinite subset in which each element is unrelated to each other element.

This section considers under what circumstances we can safely select any member from the set of minimally developed unifications and know that further consistency checking and unification with the chosen unification will yield global consistency. In
order to do this we need to impose stronger constraints on the unification set. In particular, we must ensure that unification sets possess a greatest element.

**Definition 16.** An element $X \in S$ is a greatest element of a partially ordered set, $(S, dv)$, iff $\forall X' \in S, X' dv X$. We denote such a greatest element as $g(S, dv)$. If a greatest element does not exist $g(S, dv) = \bot$.

It is worth pointing out again that we are considering uniqueness up to equivalence. Thus, in effect, the description that is generated by $g(S, dv)$ will be randomly chosen from within an equivalence class. Greatest elements are stronger than maximal elements since for greatest elements all other members of the set must be developments of the greatest element. This is not required with maximal elements for which there may exist elements that are not ancestors or descendents of a maximal element. We introduce the following obvious notation.

**Notation 2.** If it exists, we call $g(\mathcal{U}(dv_i, X_i), dv_i)$ the greatest unification.

We have a number of immediate results, proofs of these results can be found in [4].

**Proposition 11.** (i) If a greatest unification exists then all infinite chains have an upper bound.

(ii) If it exists, $g(\mathcal{U}(dv_i, X_i), dv_i) \subseteq \mathcal{U}(dv_i, X_i)$ i.e. the greatest element is a minimally developed unification.

(iii) A greatest element is unique up to equivalence.

(iv) Assuming Property 1,

$$\forall X, X' \in \mathcal{U}(dv_i, X_i), X \preceq_{ \cap dv_i } X' \iff g(\mathcal{U}(dv_i, X_i), dv_i) = \bot.$$ 

The last of these results shows that the existence of a greatest unification is the only circumstance that will yield a unique minimally developed unification, i.e. the minimally developed unification is unique up to equivalence if and only if the unification set has a greatest element.

As expected, the property that we will impose on the unification set, in order to allow us to choose any member of the set of minimally developed unifications, is that it has a greatest element, i.e. We have Property 3.

**Property 3.** If $\mathcal{U}(dv_i, X_i) \neq \emptyset$ then $g(\mathcal{U}(dv_i, X_i), dv_i) \neq \bot$.

We assume the following greatest unification function, $\mathscr{L}$; it is denoted thus, because it corresponds to the least developed unification that we have been searching for.

**Definition 17.** If $g(\mathcal{U}(dv_i, X_i), dv_i) = \bot$, then $\mathscr{L}(dv_i, X_i) = \emptyset$; otherwise

$$\mathscr{L}(dv_i, X_i) = \{g(\mathcal{U}(dv_i, X_i), dv_i)\}.$$
So, the function $\mathcal{L}$ returns the empty set if a greatest unification does not exist and a singleton set containing the greatest unification otherwise. Now we need to validate that $\mathcal{L}$ upholds (U.i) and (U.ii). These arise as immediate consequences of the next results (proofs can be found in [4]).

**Proposition 12.** Given Property 3,

(i) $\mathcal{L}^n (dv_i, X_i) \subseteq \mathcal{U}^n (dv_i, X_i)$.

(ii) $\mathcal{U}^n (dv_i, X_i) \neq \emptyset \Rightarrow \mathcal{L}^n (dv_i, X_i) \neq \emptyset$.

We can also consider the equivalent of Property 1 for $\mathcal{L}$.

**Property 4.** $X \in \mathcal{U}^n (dv_i, X_i) \Rightarrow X \cap dv_i Y$ where $Y \in \mathcal{L}^n (dv_i, X_i)$.

We can see that this property follows directly from the existence of a greatest element.

**Proposition 13.** Property 3 $\Rightarrow$ Property 4.

**Proof.** $\mathcal{U}^n (dv_i, X_i) \neq \emptyset \Rightarrow \mathcal{L}^n (dv_i, X_i) \neq \emptyset$, the result follows immediately from the definition of $\mathcal{L}$. □

We will also use the following simple result.

**Proposition 14.** Given Property 3,

$$Y \in \mathcal{L}(dv,X)(dv',X') \land Y' \in \mathcal{L}(dv,X)(dv',X')(dv'',X'') \Rightarrow Y' dv \cap dv' Y.$$

**Proof.** Clearly, $Y' \in \mathcal{U}(dv,X)(dv',X')(dv'',X'')$, but we can use Proposition 1 to get $Y' \in \mathcal{U}(dv,X)(dv',X')$ and by the definition of $\mathcal{L}$ we have $Y' dv \cap dv' Y$, as required. □

We are now in a position to relate binary consistency strategies to global consistency when greatest unifications exist. We seek an associativity result and in order to express this clearly we consider a function $\beta$ which is derived from $\mathcal{L}$. The function returns a pair, with first element the intersection of the development relations considered and second element the greatest unification. Notice a bottom element is returned as greatest unification if either a greatest unification does not exist or one of the descriptions given as an argument is undefined.

**Definition 18.** $\beta(dv,X)(dv',X') = (dv \cap dv', Y)$ where

if $X = \bot \lor X' = \bot \lor \mathcal{L}(dv,X)(dv',X') = \emptyset$ then $Y = \bot$
otherwise $Y \in \mathcal{L}(dv,X)(dv',X')$.

We will prove associativity of $\beta$ by relating the two possible binary bracketings of $\beta$ to $\mathcal{L}(dv,X)(dv',X')(dv'',X'')$. 
Proposition 15. Given Property 3, \( r(\beta(dv,X)(\beta(dv',X')(dv'',X'''))) \supseteq_{dv \cap dv' \cap dv''} Y \) where \( Y \in L'(dv,X)(dv',X')(dv'',X'') \) and \( r \) is the right projection function, which yields the second element of a pair.

Proof. Take \( Y = r(\beta(dv,X)(\beta(dv',X')(dv'',X'''))) \) and \( Y' \in L'(dv,X)(dv',X')(dv'',X'') \). By transitivity of development \( Y \in \mathcal{U}(dv,X)(dv',X')(dv'',X'') \), so by the definition of \( L' \) we get \( Y = dv \cap dv' \cap dv'' Y' \). Also, let \( Y'' = r(\beta(dv',X')(dv'',X'')) \). By, Proposition 14 \( Y' = dv \cap dv' \cap dv'' Y'' \). Also, \( Y' \in \mathcal{U}(dv,X)(dv',X')(dv'',X'') \) so \( Y' = dv X \) and therefore, \( Y' \in \mathcal{U}(dv,X)(dv' \cap dv'', Y'') \). But, \( Y \in L'(dv,X)(dv' \cap dv'', Y'') \), so, it is the greatest element in \( \mathcal{U}(dv,X)(dv' \cap dv'', Y'') \) and thus, \( Y' = dv \cap dv' \cap dv'' Y \). This gives us \( Y = dv \cap dv' \cap dv'' Y' \) and \( Y' = dv \cap dv' \cap dv'' Y \) and thus, \( Y \supseteq_{dv \cap dv' \cap dv''} Y' \), as required. □

Proposition 16. Given Property 3, \( r(\beta(\beta(dv,X)(dv',X')(dv'',X'''))) \supseteq_{dv \cap dv' \cap dv''} Y \) where \( Y \in L'(dv,X)(dv',X')(dv'',X'') \).

Proof. Similar to proof of Proposition 15. □

Now if we define equality pairwise as
\[(dv,X) = (dv',X') \text{ iff } dv = dv' \land X \supseteq_{dv \cap dv'} X'\]
the following result is straightforward.

Corollary 3. Given Property 3,
\[\beta(dv,X)(\beta(dv',X')(dv'',X'')) = \beta(\beta(dv,X)(dv',X'))(dv'',X'').\]

Proof. Follows immediately from previous two results, Propositions 15 and 16. □

This is a full associativity result which gives us that any bracketing of \( \beta(dv_1,X_1), \ldots, (dv_n,X_n) \) is equal. Since \( \beta \) is just an alternative coding of \( L \) that facilitates clarity of expression, we have full associativity of \( L \) and that a consistency strategy using \( L \) can be composed of any ordering of binary consistency checks, in particular, \( \Pi_L = C \). So, if greatest unifications exist, we can obtain global consistency from any appropriate series of binary consistency checks. This is an important result that arises from a very well behaved class of unification.

We know that the existence of a greatest unification will allow us to safely choose just one description from the minimally developed unification set. In a similar way as in Section 5.2 we generalise the condition we require to all possible unifications that can be performed in an FDT.

Property 5. An FDT, \( ft \), satisfies Property 5 iff
\[\forall X_i \in DES_{fi}, \ dv_k \in DEV_{fi}, (\mathcal{U}(dv_i,X_i) \neq \emptyset \Rightarrow g(\mathcal{U}(dv_i,X_i), \cap dv_i) \neq \perp).\]
This property ensures that any possible combination of descriptions and development relations in \( \mathcal{F} \) will generate a unification set with a greatest element. Satisfaction of this property will guarantee that we can always safely select just one element from the minimally developed unification set.

7. Strategies for checking balanced consistency

The majority of work to be found in the literature on consistency has addressed more restricted classes of consistency than we have considered. In particular, to date, balanced consistency has almost exclusively been focused on. So, what happens to the theory considered so far in these circumstances? This section then restricts itself to balanced intra language consistency and \( dv \) a preorder.

We have a number of preparatory definitions. The following is the standard set theoretic notion of a lower bound of a set.

**Definition 19.** \( X \in \text{DES}_\mathcal{F} \) is a lower bound of \( Z \subseteq \text{DES}_\mathcal{F} \) iff \( \forall X' \in Z, X < dv X' \). The set of all lower bounds of \( Z \) is denoted \( \text{lb}(Z, dv) \). If a lower bound does not exist \( \text{lb}(Z, dv) = \emptyset \).

A lower bound of \( Z \) is a development of all elements of \( Z \). Notice a lower bound does not have to be a member of \( Z \) in contrast to a greatest element. It should be clear that for balanced consistency lower bounds correspond to unifications, i.e. \( \forall_{dv}(X_1, \ldots, X_n) = \text{lb}(\{X_1, \ldots, X_n\}, dv) \). In particular, the fact that the ordering of descriptions in balanced unification is unimportant is reflected by the descriptions being interpreted as a set in \( \text{lb} \).

In standard fashion we can also define the concept of a greatest lower bound.

**Definition 20.** For \( Z \subseteq \text{DES}_\mathcal{F} \) \( \text{glb}(Z, dv) \) is a lower bound such that all other lower bounds are a development of \( \text{glb}(Z, dv) \), i.e. \( \text{glb}(Z, dv) \in \text{lb}(Z, dv) \land (\forall X \in \text{lb}(Z, dv), X < dv \text{glb}(Z, dv)) \). If a greatest lower bound does not exist \( \text{glb}(Z, dv) = \bot \).

It should again be clear that a greatest lower bound of a set of descriptions is a greatest unification of the descriptions. In particular, note that the ordering of the unification set by \( \cap dv \) in the general (unbalanced) case has been collapsed to just \( dv \).

We can now define consistency in this restricted setting.

**Definition 21.** \( C_{dv}(X_1, \ldots, X_n) \iff \exists X \in \text{lb}(\{X_1, \ldots, X_n\}, dv) \text{ s.t. } \Psi(X) \).

With this theory we can also simply characterise when all descriptions in an FDT are balanced consistent by \( dv \), i.e. the FDT is *completely consistent* by \( dv \).

**Proposition 17.** \( \forall Z \subseteq \text{DES}_\mathcal{F} \land dv \in \text{DEV}_\mathcal{F}, \exists X \in \text{lb}(Z, dv) \land \Psi(X) \Rightarrow \forall X_1, \ldots, X_n \in \text{DES}_\mathcal{F}, C_{dv}(X_1, \ldots, X_n) \text{ holds.} \)
Proof. Straightforward. □

I.e. if all subsets of $DES_{fi}$ have a lower bound then all specifications are consistent by $dv$.

An alternative check for complete consistency is that an internally valid terminal element exists for $dv$. A development relation $dv$ has a terminal or bottom element, denoted $\perp_{dv}$, if and only if $\forall X \in DES_{fi}. \perp_{dv} dv X$.

**Proposition 18.** $DES_{fi}$ has an internally valid bottom element $\Rightarrow \forall X_1, \ldots, X_n \in DES_{fi}. C_{dv}(X_1, \ldots, X_n)$ holds.

Proof. Immediate. □

**Example 5.** As a simple illustration, for LOTOS, $C_{\leq tr}$ and $C_{ext}$ are completely consistent, since all groups of specifications have common refinements. For example, the process stop, which offers only the empty trace is a terminal element for $\leq_{tr}$ and the process that offers a choice of all possible actions at all points in the computation is a terminal element for $ext$.

What, in this restricted setting, enables us to obtain global consistency from binary consistency? We would like to locate an equivalent of the existence of greatest unifications. As indicated earlier, the greatest lower bound gives us this equivalent.

**Proposition 19.** $glb\{X_1, \ldots, X_n\}, dv\} \neq \perp \Rightarrow glb\{X_1, \ldots, X_n\}, dv\} \in L_{dv}(X_1, \ldots, X_n)$.

Proof. By definition. □

So, the property that we require for balanced consistency checking to be performed incrementally is the following.

**Property 6.** $\forall \{X_1, \ldots, X_n\} \subseteq DES_{fi} \land \forall dv \in DEV_{fi}, lb\{X_1, \ldots, X_n\}, dv\} \neq \emptyset \Rightarrow glb\{X_1, \ldots, X_n\}, dv\} \neq \perp$.

This property ensures that if a lower bound exists then a greatest lower bound can be found, i.e. the unification of $X_1, \ldots, X_n$ is non-empty implies a greatest unification exists. It is clear from the theory of greatest unifications we have presented and from set theory that taking greatest lower bounds is associative, i.e.

$$glb\{glb\{X_1, X_2\}, dv\}, X_3\}, dv\} = glb\{X_1, glb\{X_2, X_3\}, dv\}, dv\}.$$

With these concepts we can identify what is the most well behaved class of development.

**Definition 22.** $(DES_{fi}, dv)$ is cocomplete iff $\forall S \subseteq DES_{fi}, glb(S, dv) \neq \perp$.

Cocompleteness is related to the standard concept of a complete partial order, see for example [25], which considers the existence of least upper bounds as opposed to
greatest lower bounds in our framework. If development is cocomplete for a particular FDT according to a development relation then all specifications are balanced consistent and we can adopt any relevant incremental consistency checking strategy. All descriptions are consistent since a lower bound exists for all collections of descriptions and incremental consistency checking strategies are well behaved since a single greatest unification always exists.

**Example 6.** It is pleasing to discover that for a number of the LOTOS relations balanced consistency is indeed cocomplete. It is beyond the scope of this paper to consider the unification strategies that yield these results, but we can point the reader to [23] where a number of the main results were verified and to [9] which extends these results. We simply reproduce the main results here:

(\text{DES}_{\text{LOTOS}}, \text{ext}) \text{ and } (\text{DES}_{\text{LOTOS}}, \leq_{\nu}) \text{ are cocomplete;}

(\text{DES}_{\text{LOTOS}}, \text{red}) \text{ is not cocomplete.}

The lack of cocompleteness for reduction arises because it is not completely consistent. However, for any set, \( S \) say, of consistent specifications under reduction, there is a greatest lower bound, i.e. \( \text{glb}(S, \text{red}) \neq \bot \).

Thus, in the case of balanced consistency the set of LOTOS preorders are relatively well behaved.

8. Concluding remarks

This paper has presented a general interpretation of consistency for multiple viewpoint models of system development and investigated possible consistency checking strategies. Our interpretation of consistency is extremely broad, embracing intra and inter language consistency, balanced and unbalanced consistency and both binary and global consistency. This generality arises as a direct consequence of the requirements of viewpoints modelling in Open Distributed Processing.

The main original contribution of this paper is the investigation of possible strategies for consistency checking. These address the issue of obtaining global consistency

<table>
<thead>
<tr>
<th>Class of consistency</th>
<th>Implications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unbalanced Inter lang.</td>
<td>Unbounded inf. chains</td>
</tr>
<tr>
<td>Unbalanced Intra lang.</td>
<td>Unbounded inf. chains</td>
</tr>
<tr>
<td>Unbalanced Intra lang.</td>
<td>Bounded inf. chains</td>
</tr>
<tr>
<td>Unbalanced Intra lang.</td>
<td>Greatest unifs.</td>
</tr>
<tr>
<td>Balanced Intra lang.</td>
<td>Unbounded inf. chains</td>
</tr>
<tr>
<td>Balanced Intra lang.</td>
<td>Bounded inf. chains without glb's</td>
</tr>
<tr>
<td>Balanced Intra lang.</td>
<td>glbs always exist</td>
</tr>
<tr>
<td>Balanced Intra lang.</td>
<td>Cocomplete</td>
</tr>
</tbody>
</table>
incrementally through a series of, possibly binary, consistency checks; thus, enabling
global consistency to be deduced from a number of smaller consistency checks. This
topic has been investigated in the past, but only in the context of a restricted class
of consistency. In particular, this is the first paper to investigate consistency checking
strategies for as general an interpretation of consistency as ours. The main difference
between our theory and earlier work is that we handle unbalanced consistency.

As a reflection of our general handling of consistency a spectrum of classes of
consistency checking have been identified. These range from the very poorly behaved
to the very well behaved. These classes are summarised in Table 2.

In general, the consistency problem is more straightforward and well behaved the
further down the table you go.

Acknowledgements

We would like to thank the anonymous reviewers of this paper. In particular,
reviewer #2 made a number of valuable suggestions which have improved the quality
of the paper.

Appendix A

The appendices of this paper collect together a number of results that we have not
had room to consider in the main text.

Proofs from Section 4.2

Proposition A.1. (i) $\mathcal{U}_{te}(S_1, \mathcal{U}_{te}(S_2, S_3)) = \mathcal{U}_{te}(\mathcal{U}_{te}(S_1, S_2), S_3)$.
(ii) $U(S_1, U(S_2, S_3)) = U(U(S_1, S_2), S_3)$ where $U$ is the conf unification operator.

Proof. (i) Notice that since $te$ is an equivalence, $\mathcal{U}_{te}$ defines a single specification (up
to equivalence) and thus, we can abuse notation and use $\mathcal{U}_{te}$ as if it returns a single
specification rather than a singleton set.

Furthermore, we define $\mathcal{U}_{te}$ as

$$\mathcal{U}_{te}(P_1, P_2) = \begin{cases} P_1 \text{ if } P_1 \text{ te } P_2 \text{ then } P_1 \text{ else } \omega \end{cases}$$

where $\omega$ is characterised as $Tr(\omega) = \emptyset$ and $\forall \sigma \in Act^* \cdot Ref((\omega, \sigma)) = \emptyset$. $\omega$ is an imaginary
process, in the style of [23], which has a fundamentally contradictory behaviour, i.e. it
neither offers or refuses any action. If the unification returns $\omega$ then the specifications
were not consistent.

Now it is easy to see that

$$\mathcal{U}_{te}(S_1, \mathcal{U}_{te}(S_2, S_3)) = \begin{cases} S_1 \text{ te } S_2 \text{ te } S_3 \text{ then } S_1 \text{ else } \omega = \mathcal{U}_{te}(\mathcal{U}_{te}(S_1, S_2), S_3) \end{cases}.$$
(ii) Define $U(P_1, P_2)$ as

\[
\text{Tr}(U) = \text{Tr}(P_1) \cap \text{Tr}(P_2) \quad \text{and} \\
\forall \sigma \in \text{Tr}(U). \text{Ref}(U, \sigma) = \text{Ref}(P_1, \sigma) \cap \text{Ref}(P_2, \sigma).
\]

Other definitions of conf unification can be given, but this is the most well behaved. It is a simple task to take this definition and show that

\[
U(S_1, U(S_2, S_3)) = U(U(S_1, S_2), S_3)
\]
as required. $\square$

Proofs from Section 5

**Proposition A.2.** Given Property 1,

\[
\exists X \in \mathcal{U}(dv_1, X_1) \ldots (dv_m, X_m) \Rightarrow \Pi_{-\#}(dv_1, X_1) \ldots (dv_m, X_m)
\]

where

\[
\Pi_{-\#}(dv_1, X_1) \ldots (dv_m, X_m)
\]

\[
= ((\exists Y_1 \in \mathcal{M}(dv_1, X_1)(dv_2, X_2) \land X \in \mathcal{U}(dv_1, X_1)(dv_2, X_2)) \land \\
(\exists Y_2 \in \mathcal{M}(dv_1 \cap dv_2, Y_2)(dv_3, X_3) \land X \in \mathcal{U}(dv_1 \cap dv_2, Y_2)(dv_3, X_3)) \land \\
(\exists Y_3 \in \mathcal{M}(dv_1 \cap dv_2 \cap dv_3, Y_2)(dv_4, X_4) \land X \in \mathcal{U}(dv_1 \cap dv_2 \cap dv_3, Y_2)
\]

\[
(dv_4, X_4)) \land \\
\ldots \\
\ldots \\
(\exists Y_{m-2} \in \mathcal{M}(dv_1 \cap \ldots \cap dv_{m-2}, Y_{m-3})(dv_{m-1}, X_{m-1})
\]

\[
\land X \in \mathcal{U}(dv_1 \cap \ldots \cap dv_{m-2}, Y_{m-3})(dv_{m-1}, X_{m-1})) \land \\
(\exists Y_{m-1} \in \mathcal{M}(dv_1 \cap \ldots \cap dv_{m-2}, Y_{m-2})(dv_m, X_m)
\]

\[
\land X \in \mathcal{U}(dv_1 \cap \ldots \cap dv_{m-2}, Y_{m-2})(dv_m, X_m))
\]

Notice that we are not considering $\Pi$ directly, rather we consider the unification strategy $\Pi$ which adds a second condition on every step of the algorithm. This condition states that $X$, the original unification, is in the unification set relevant to that step. Carrying this condition will simplify the induction proof that we perform and clearly gives us a stronger result than we actually need.

**Proof.** We prove this result using induction on the number of descriptions (and hence development relations) that are considered, i.e. induction on $m$ above. We will prove two base cases in order to indicate the pattern of the proof. This pattern is reflected in the proof of the inductive step.

**Base case 1, $m = 2$.** Notice $m = 1$ does not exist (although a trivial formulation could be given). We wish to prove

\[
(\text{As}) \exists X \in \mathcal{U}(dv_1, X_1)(dv_2, X_2) \Rightarrow ((a) \exists Y_1 \in \mathcal{M}(dv_1, X_1)(dv_2, X_2) \land \\
(b) X \in \mathcal{U}(dv_1, X_1)(dv_2, X_2)).
\]
This is straightforward. Firstly, (b) follows immediately from our assumption, (As), then (a) is a direct consequence of (b) from (U.ii).

**Base case 2, m = 3.** We wish to prove

\[
\text{(As)} \quad \exists X \in \mathcal{U}(d_{v_1}, X_1)(d_{v_2}, X_2)(d_{v_3}, X_3) \Rightarrow
\]

(a) \( \forall Y_1 \in \mathcal{M}(d_{v_1}, X_1)(d_{v_2}, X_2) \)

(b) \( X \in \mathcal{U}(d_{v_1}, X_1)(d_{v_2}, X_2) \)

(c) \( \exists Y_2 \in \mathcal{M}(d_{v_1} \cap d_{v_2}, Y_1)(d_{v_3}, X_3) \)

(d) \( X \in \mathcal{U}(d_{v_1} \cap d_{v_2}, Y_1)(d_{v_3}, X_3))

Firstly, by observing that from Proposition 1, \( X \in \mathcal{U}(d_{v_1}, X_1)(d_{v_2}, X_2)(d_{v_3}, X_3) \) implies that \( X \in \mathcal{U}(d_{v_1}, X_1)(d_{v_2}, X_2) \) we can reproduce the argument of base case 1 to obtain (a) and (b).

Now from (a) and (b) we can use Property 1 to get \( \exists Y_1' \in \mathcal{M}(d_{v_1}, X_1)(d_{v_2}, X_2) \) such that \( X \in \mathcal{U}(d_{v_1} \cap d_{v_2}, Y_1') \) and from \( X \in \mathcal{U}(d_{v_3}, X_3) \) from our assumption, (As), we have \( X \in \mathcal{U}(d_{v_1} \cap d_{v_2}, Y_1')(d_{v_3}, X_3) \) which gives us (d) and then we can use (U.ii) to get \( \exists Y_2 \in \mathcal{M}(d_{v_1} \cap d_{v_2}, Y_1')(d_{v_3}, X_3) \), i.e. (c). This completes the verification of base case 2.

**Inductive step.** We wish to prove that: Proposition (1) \( \Rightarrow \) proposition (2), where.

Proposition (1) states:

\[
(\text{As.i}) \quad \exists X \in \mathcal{U}(d_{v_1}, X_1) \ldots (d_{v_n}, X_n) \Rightarrow
\]

\[
\begin{align*}
(1.1) & \quad \exists Y_1 \in \mathcal{M}(d_{v_1}, X_1)(d_{v_2}, X_2) \land X \in \mathcal{U}(d_{v_1}, X_1)(d_{v_2}, X_2) \land \\
(1.2) & \quad \exists Y_2 \in \mathcal{M}(d_{v_1} \cap d_{v_2}, Y_1)(d_{v_3}, X_3) \land X \in \mathcal{U}(d_{v_1} \cap d_{v_2}, Y_1)(d_{v_3}, X_3) \land \\
& \quad \ldots \\
& \quad \ldots \\
& \quad (1.n - 2) \quad \exists Y_{n-2} \in \mathcal{M}(d_{v_1} \cap \ldots \cap d_{v_n-2}, Y_{n-3})(d_{v_{n-1}}, X_{n-1}) \\
& \quad \land X \in \mathcal{U}(d_{v_1} \cap \ldots \cap d_{v_n-2}, Y_{n-3})(d_{v_{n-1}}, X_{n-1}) \land \\
& \quad (1.n - 1) \quad \exists Y_{n-1} \in \mathcal{M}(d_{v_1} \cap \ldots \cap d_{v_n-1}, Y_{n-2})(d_{v_n}, X_n) \\
& \quad \land X \in \mathcal{U}(d_{v_1} \cap \ldots \cap d_{v_n-1}, Y_{n-2})(d_{v_n}, X_n) \\
\end{align*}
\]

Proposition (2) states:

\[
(\text{As.ii}) \quad \exists X \in \mathcal{U}(d_{v_1}, X_1) \ldots (d_{v_{n+1}}, X_{n+1}) \Rightarrow
\]

\[
\begin{align*}
(2.1) & \quad \exists Y_1 \in \mathcal{M}(d_{v_1}, X_1)(d_{v_2}, X_2) \land X \in \mathcal{U}(d_{v_1}, X_1)(d_{v_2}, X_2) \land \\
(2.2) & \quad \exists Y_2 \in \mathcal{M}(d_{v_1} \cap d_{v_2}, Y_1)(d_{v_3}, X_3) \land X \in \mathcal{U}(d_{v_1} \cap d_{v_2}, Y_1)(d_{v_3}, X_3) \land \\
& \quad \ldots \\
& \quad \ldots \\
& \quad (2.n - 2) \quad \exists Y_{n-2} \in \mathcal{M}(d_{v_1} \cap \ldots \cap d_{v_n-2}, Y_{n-3})(d_{v_{n-1}}, X_{n-1}) \\
& \quad \land X \in \mathcal{U}(d_{v_1} \cap \ldots \cap d_{v_n-2}, Y_{n-3})(d_{v_{n-1}}, X_{n-1}) \land \\
& \quad (2.n - 1) \quad \exists Y_{n-1} \in \mathcal{M}(d_{v_1} \cap \ldots \cap d_{v_n-1}, Y_{n-2})(d_{v_n}, X_n) \\
& \quad \land X \in \mathcal{U}(d_{v_1} \cap \ldots \cap d_{v_n-1}, Y_{n-2})(d_{v_n}, X_n) \land \\
& \quad (2.n) \quad \exists Y_n \in \mathcal{M}(d_{v_1} \cap \ldots \cap d_{v_n}, Y_{n-1})(d_{v_{n+1}}, X_{n+1}) \\
& \quad \land X \in \mathcal{U}(d_{v_1} \cap \ldots \cap d_{v_n}, Y_{n-1})(d_{v_{n+1}}, X_{n+1}) \\
\end{align*}
\]
So, assume Proposition (1). It is clear that the first \( n - 1 \) steps of Proposition (2), i.e. (2.1), (2.2), \ldots, (2.n - 2), (2.n - 1), can be obtained directly from Proposition (1). So, we need that Proposition (1) and assumption (As.ii) imply (2.n). We know, \( \exists Y_{n-1} \in \mathcal{U}(d_{v_1} \cap \cdots \cap d_{v_{n-1}}, Y_{n-2})(d_{v_n}, X_n) \) and \( X \in \mathcal{U}(d_{v_1} \cap \cdots \cap d_{v_{n-1}}, Y_{n-2}) \) \( (d_{v_n}, X_n) \) from (1.n - 1), so we can use Property 1 to get that \( \exists Y'_{n-1} \in \mathcal{U}(d_{v_1} \cap \cdots \cap d_{v_{n-1}}, Y_{n-2})(d_{v_n}, X_n) \) such that \( X' \in \mathcal{U}(d_{v_1} \cap \cdots \cap d_{v_{n-1}}, Y'_{n-1}) \) \( (d_{v_n+1}, X_{n+1}) \) since \( X \in \mathcal{U}(d_{v_1} \cap \cdots \cap d_{v_{n-1}}, Y_{n-1}) \). This gives us the second half of (2.n) and the first half follows directly from (U.ii).

By induction, the result follows. \( \square \)

**Proofs from Section 5.2**

**Proposition A.3.** \((\text{DESLO}, (\leq))\) satisfies Property 1:

\[ X \in \mathcal{U} (X_1, \ldots, X_n) \Rightarrow \exists X' \in \mathcal{U} (X_1, \ldots, X_n) \text{ s.t. } X \leq X'. \]

**Proof.** Assume \( \{P_0, P_1, P_2, \ldots\} \) is an infinitely increasing chain. Thus, \( P_i \leq P_{i+1} \) for all \( i \in \mathbb{N} \). We show that we can construct an upper bound for this chain.

1. If all the \( P_i \)'s are trace finite then construct \( P \) such that

   \[ \text{Tr}(P) = \bigcap_i \text{Tr}(P_i) \]

   \[ \forall \sigma \in \text{Tr}(P). \text{Ref}(P, \sigma) = \bigcup_i \text{Ref}(P_i, \sigma). \]

   Clearly \( P \) will be trace finite, thus, \( P_i \sim P \).

   In addition, the construction of \( P \) ensures that \( P_i \text{conf} P \) and since \( \text{conf} \subseteq \text{conf}^* \) we know that for all \( P_i \), \( P_i \leq P \) thus, \( \{P_0, P_1, P_2, \ldots\} \) has an upper bound.

2. If all the \( P_i \)'s are trace infinite then construct \( P \) such that

   \[ \text{Tr}(P) = \bigcup_i \text{Tr}(P_i) \]

   \[ \forall \sigma \subset \text{Tr}(P). \text{Ref}(P, \sigma) = \bigcup_i \text{Ref}(P_i, \sigma) \]

   Clearly, \( P \) will be trace infinite, thus, \( P_i \sim P \).

   In addition, \( P_i \text{conf} P \) since,

   \[ \forall \sigma \in \text{Tr}(P). \]

   \[ \sigma \in \text{Tr}(P_i) \Rightarrow \text{Ref}(P_i, \sigma) \subseteq \bigcup_i \text{Ref}(P_i) = \text{Ref}(P, \sigma) \land \]

   \[ \sigma \notin \text{Tr}(P_i) \Rightarrow \text{Ref}(P_i, \sigma) = \emptyset \subseteq \text{Ref}(P, \sigma) \]

   and since \( \text{conf} \subseteq \text{conf}^* \) for all \( P_i \), \( P_i \leq P \) thus, \( \{P_0, P_1, P_2, \ldots\} \) has an upper bound.
3. We cannot have a chain which contains a mixture of finite and infinite specifications because then $\leq$ could not hold between them.

This completes the proof, as required. $\square$

References


