Strategies for Consistency Checking, the Choice of Unification *

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

Computing Laboratory, University of Kent, Canterbury, CT2 7NF, UK.
(Phone: + 44 1227 764000, Fax 44 1227 762811
Email: {H.Bowman,E.A.Boiten,J.Derrick,mwas}@ukc.ac.uk.)

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. Our work particularly focuses on the use of viewpoints in Open Distributed Processing (ODP) which is an ISO/ITU standardisation framework. 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 in ODP. 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.
Keywords: Viewpoints, Consistency, ODP, 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 [10] [15] [1] [11]. 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 targetted 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 [14] is being proposed for the information viewpoint and LOTOS [3] for the computational viewpoint.

Figure 1 [7] 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 realized 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.

*This work was partially funded by British Telecom Research Labs., Martlesham, Ipswich, U.K. and the Engineering and Physical Sciences Research Council under grant number GR/K13035.

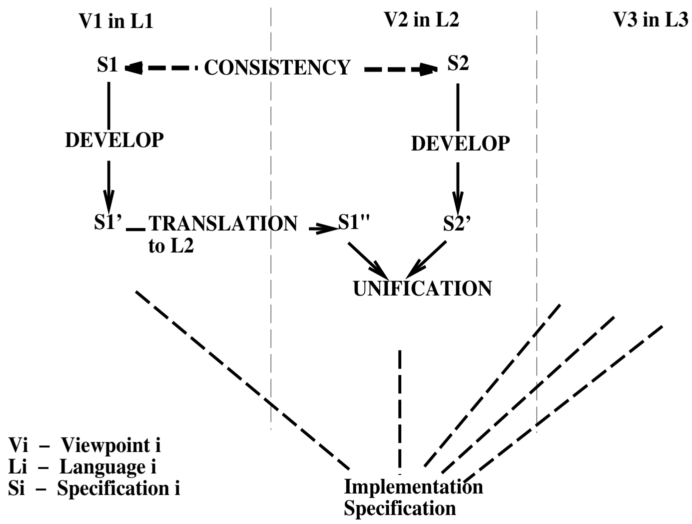

Figure 1: Relating Viewpoints

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 [5]. 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 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] [17], which has universally looked at, what we call, *balanced consistency*.

We have considered viewpoint consistency for ODP in a number of papers [5] [9] [8] [16] 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 least developed unification does not exist and when such a unique 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 and ZF set theory.

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 [12] such as the LOTOS conformance relation **conf**. 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.

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 seems necessary in order to obtain a rich enough theory. Furthermore, this paper is motivated by the search for incremental development strategies and transitivity of development seems a prerequisite of such incremental evolution of specifications. In particular without transitivity, we may develop a specification A into a specification B and then evolve B into C and find that C is not a development of A. So, this paper assumes transitivity and reflexivity of the development relations used, i.e. they are preorders.

We will need to talk about sets of possible developments of a specification. Thus, we introduce the following notation:

Definition 1 *For $X \in DES$ and $dv \in DEV$,*

$$D(X, dv) = \{X' \ :\ X'\ dv\ X\}.$$

So, $D(X, dv)$ is the set of all developments of X by dv.

We must also consider what interpretation of equivalence (which we denote $\asymp$) we should adopt. A natural, and standard, interpretation is:-

$$X \asymp_{dv} X' \text{ iff } X\ dv\ X' \wedge X'\ dv\ X$$

Thus, two descriptions are equivalent if and only if they are both developments of the other. With transitivity of dv this interpretation gives us that two specifications in any cycle by the relation dv are equivalent. $\asymp_{dv}$ will play the role of identity in our theory. The following results can be easily determined (see [4] for proofs):

Proposition 1
(i) $\asymp_{dv}$ *is an equivalence.*
(ii) dv *is a partial order with identity* $\asymp_{dv}$.

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 $\asymp_{dv}$. 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 $\asymp_{dv}$ factored out.

Definition 2
Overlining is an operation that can be applied to an arbitrary partial order, dv, with the following effect:-

$$\overline{dv} = dv \backslash \asymp_{dv}.$$

$\overline{dv}$ enables us to consider directly the part of dv that excludes identical descriptions by $\asymp_{dv}$. $\overline{dv}$ is strict with regard to dv in the same way that $\subset$ is strict with regard to $\subseteq$. Note in particular 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_{ft}, the set of all development relations as DEV_{ft} and the set of all semantic maps as SEM_{ft}.

Basic Definition. In its general form consistency is a check which takes any number of descriptions, $X_1, X_2, ..., X_n$, and returns true if all the descriptions are consistent and false otherwise. This check will be performed according to a group of development relations, $dv_1, dv_2, ..., dv_n$, one per description, and is denoted, $C(dv_1, X_1)(dv_2, X_2)...(dv_n, X_n)$, a shorthand for which is $\overset{1..n}{C}(dv_i, X_i)$. 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 $X_1, X_2, ..., X_n$ 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, $dv_1, dv_2, ..., dv_n$. Definition 3 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 figure 2 and is formalized in definition 3. We denote this interpretation of consistency as C.

Definition 3 (Consistency)
$\overset{1..n}{C}(dv_i, X_i)$ *holds, iff* $\exists X \in DES$ *s.t.* $(X \;\; dv_1 \; X_1 \;\; \wedge ... \wedge \;\; X \; dv_n \; X_n) \;\wedge \Psi(X)$.

The internal validity check in the above definition formalizes 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, Ψ 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. a Z specification that contains an operation $[n! : \mathbb{N} | n! = 5 \wedge n! = 3]$ has no real implementation. This ensures that definition 3 in the case that n=1 coincides with what is commonly called "consistency" of a single specification.

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

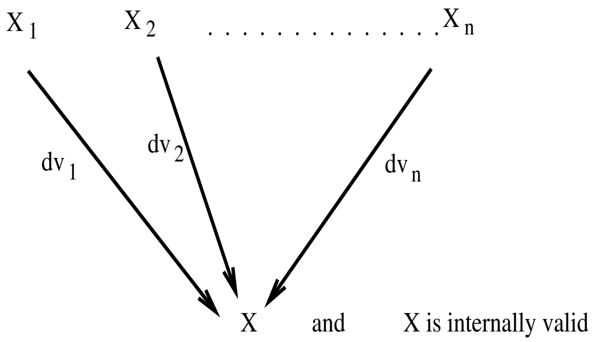

Figure 2: A Consistency Check

Definition 4 (Unification Set)
$\mathcal{U}(dv_1, X_1)(dv_2, X_2)...(dv_n, X_n) = \{X \in DES \ : \ X \ dv_1 \ X_1 \ \wedge ... \wedge \ X \ dv_n \ X_n\}$.

We will use the notation $\overset{1..n}{\mathcal{U}}(dv_i, X_i)$ as a shorthand for $\mathcal{U}(dv_1, X_1)(dv_2, X_2)...(dv_n, X_n)$. The unification set is the set of all common developments of a list of descriptions, i.e. the set of all unifications. Clearly, $\overset{1..n}{C}(dv_i, X_i)$ 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 [8].

The following result can be immediately observed.

Proposition 2
$\{(dv_1, X_1), ..., (dv_n, X_n)\} \supseteq \{(dv'_1, X'_1), ..., (dv'_m, X'_m)\} \Longrightarrow \mathcal{U}(dv_1, X_1)...(dv_n, X_n) \subseteq \mathcal{U}(dv'_1, X'_1)...(dv'_m, X'_m)$

Proof
If $\overset{1..n}{\mathcal{U}}(dv_i, X_i) = \emptyset$ the result follows trivially. So, take $X \in \overset{1..n}{\mathcal{U}}(dv_i, X_i)$ i.e. $X \ dv_1 \ X_1 \ \wedge ... \wedge \ X \ dv_n \ X_n$. For any $X'_j \ s.t. \ 1 \leq j \leq m, \ X \ dv'_j \ X'_j$ since by the hypothesis $\exists X_i$ (for $1 \leq i \leq n$) such that $dv_i = dv'_j$ and $X_i = X'_j$. So, $X \in \mathcal{U}(dv'_1, X'_1)...(dv'_m, X'_m)$, as required. □

This proposition expresses the obvious result that a unification of n specifications is a unification of a subset of the n specifications. An immediate corollary of proposition 2 is:

Corollary 1
$\overset{1..n}{\mathcal{U}}(dv_i, X_i) \subseteq \mathcal{U}(dv_i, X_i)...(dv_j, X_j) \ for \ 1 \leq i, j \leq n$.

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 $X_1, X_2, ..., X_n$ in the above definition will all be specifications, however, X will commonly be a semantic representation. In particular, if some of $X_1, X_2, ..., X_n$ 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 $i \neq j$.

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, thus, reviewed in the following subsections.

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:

Definition 5 Complete Consistency
A formal description technique, ft, is completely consistent according to a group of development relations $dv_1, ..., dv_n$ if and only if for all $X_1, ..., X_n$ in DES_{ft}, $\overset{1..n}{C}(dv_i, X_i)$.

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 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:-

Notation 1 (Implementation Complete)
A formal technique ft is called implementation complete iff $\forall X \in DES_{ft},\ \Psi(X)$.

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 6 (Pairwise Consistency) *Descriptions $X_1, X_2, ..., X_n$ are pairwise consistent according to development relations $dv_1, dv_2, ...dv_n$ iff $\forall i, j$ s.t. $1 \leq i, j \leq n,\ X_i\ C_{dv_i, dv_j}\ X_j$.*

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

Proposition 3
(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. $(X\ dv_1\ X_1\ \wedge\ X\ dv_2\ X_2\ \wedge ... \wedge\ X\ dv_n\ X_n)\ \wedge \Psi(X)$. Now clearly $X_i\ C_{dv_i, dv_j}\ X_j$ 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: $S_1 = [x!, y! : \mathbb{N} | x! = y!]$, $S_2 = [x!, z! : \mathbb{N} | x! = z!]$ and $S_3 = [z!, y! : \mathbb{N} | z! \neq y!]$. Intuitively these are balanced pairwise consistent, i.e. $S_1\ C\ S_2$, $S_2\ C\ S_3$, $S_1\ C\ S_3$, but, they are not globally consistent. □

Intuitively, the second part of the above proposition arises because pairwise consistency only requires the existence of a common development for each of the consistuent 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.

Balanced Consistency. Balanced consistency reflects the situation in which the specifications being checked for consistency are at the same level of abstraction; balanced consistency is written: $\overset{1..n}{C}{}_{dv} X_i$.

Definition 7 (Balanced Consistency)
$\overset{1..n}{C}(dv_i, X_i)$, is balanced iff $dv_i = dv_j,\ \forall i, j$ s.t. $1 \leq i, j \leq n$.

Once again we can consider the special case of binary balanced consistency, $X_1\ C_{dv}(X_1, X_2)\ X_2$, which is often written as C_{dv}.

Notation	Meaning
$P \xrightarrow{\mu} P'$	denotes a transition, i.e. P can do μ and consequently behaves as P'.
$\stackrel{\epsilon}{\Longrightarrow}$	reflexive and transitive closure of $\xrightarrow{i}$
$P \stackrel{\alpha\sigma}{\Longrightarrow} P'$	$\exists Q, Q' \cdot P \stackrel{\epsilon}{\Longrightarrow} Q \xrightarrow{\alpha} Q' \stackrel{\sigma}{\Longrightarrow} P'$
$P \stackrel{\sigma}{\Longrightarrow}$	$\exists P' \cdot P \stackrel{\sigma}{\Longrightarrow} P'$
$P \stackrel{\sigma}{\not\Longrightarrow}$	$\nexists P' \cdot P \stackrel{\sigma}{\Longrightarrow} P'$

Table 1: Derived transition denotations

3 Background on LOTOS

Subsequent sectios 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 α to range over $\mathcal{L}$. Furthermore, $\mathcal{L}^*$ denotes strings (or traces) over $\mathcal{L}$. The constant $\epsilon \in \mathcal{L}^*$ denotes the empty string, and the variable σ 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:

$Tr(P) = \{\sigma \in \mathcal{L}^* \mid \mathcal{P} \stackrel{\sigma}{\Longrightarrow}\}$, denotes the set of traces of a process P.
$P\ after\ \sigma = \{P' \mid P \stackrel{\sigma}{\Longrightarrow} P'\}$, denotes the set of all states reachable from P by the trace σ.
$Ref(P, \sigma) = \{X \mid \exists P' \in (P\ after\ \sigma),\ \text{s.t.}\ \forall \alpha \in X : P' \stackrel{\alpha}{\not\Longrightarrow}\}$, denotes the refusals of $P\ after\ \sigma$.

Trace Preorder. An important category of system properties that one would like have satisfied, are safety properties. 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 σ_2, it also holds for all its prefixes. In particular, all safety properties hold for the empty trace ϵ.

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 8 (trace preorder)
Given two process specifications P and Q, then P is a trace refinement of Q, denoted $P \leq_{tr} Q$, iff $Tr(P) \subseteq Tr(Q)$

Reduction. In addition to safety properties the liveness (or deadlock) properties 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, **red**, defined in [6].

Definition 9 (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 [6] 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 10 (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. $\forall \sigma \in Tr(Q),\ Ref(P,\sigma) \subseteq Ref(Q,\sigma)$

Testing Equivalence. A standard interpretation of equivalence is given by the testing equivalence relation. In particular, notice that P **te** $Q \iff P$ **red**$\cap$**red**$^{-1}$ $Q \iff P$ **ext**$\cap$**ext**$^{-1}$ Q. So, testing equivalence is a sensible identity in the sense of $\asymp$ for both **red** and **ext**.

Definition 11 (testing equivalence)
P **te** Q *iff* $Tr(P) = Tr(Q)\ \wedge\ \forall \sigma \in Tr(P),\ Ref(P,\sigma) = Ref(Q,\sigma)$.

Internal Validity. At least theoretically, we can view all LOTOS specifications as implementable. 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
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 $X_1, X_2, ..., X_n$ are consistent if, firstly, the set of possible unifications $\overset{1..n}{\mathcal{U}}(dv_i, X_i)$ 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. In particular, we would like to select just one description from the set of unifications. This would enable an incremental development strategy in which a group of viewpoints are unified and then this unification is further composed with another group of viewpoints. This situation amounts to obtaining 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 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 subsection 4.1. Then general formats for binary consistency checking are considered in section 4.2 and the central issue of least developed unification is discussed in 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 12 $X \in \overset{1..n}{\mathcal{U}}(dv_i, X_i)$ *is a representative unification iff* $(\exists X' \in \overset{1..n}{\mathcal{U}}(dv_i, X_i)\ s.t.\ \Psi(X')) \implies \Psi(X)$.

The following result is very straightforward:-

Proposition 5
ft is implementation complete and $X_1, ..., X_n \in DES_{ft} \implies \forall X \in \overset{1..n}{\mathcal{U}}(dv_i, X_i)$ X *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 and it would be a major flaw in the strategy if it did not. 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 3. However, a combination of binary consistency checks and binary unification of the form shown in figure 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 figure 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 is almost certain to be devoid of high level specification structure (e.g. operators such as |[]| in LOTOS are expanded out) [16]. 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 Π_U, where U is a particular binary unification function. Π incorporates a series of binary consistency checks, each of which uses U to perform the binary unification. U has the general form,

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

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 Π_U. Thus, Π_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 13

$$\begin{aligned}
&\Pi_U(dv_1, X_1)....(dv_n, X_n) \ \stackrel{def}{=} \\
&\quad ((\exists Y_1 \in U(dv_1, X_1)(dv_2, X_2) \ \wedge \ \Psi(Y_1)) \ \wedge && \textit{- Step 1} \\
&\quad (\exists Y_2 \in U(dv_1 \cap dv_2, Y_1)(dv_3, X_3) \ \wedge \ \Psi(Y_2)) \ \wedge && \textit{- Step 2} \\
&\quad (\exists Y_3 \in U(dv_1 \cap dv_2 \cap dv_3, Y_2)(dv_4, X_4) \ \wedge \ \Psi(Y_3)) \ \wedge && \textit{- Step 3} \\
&\qquad\qquad \dots \\
&\qquad\qquad \dots \\
&\quad (\exists Y_{n-2} \in U(dv_1 \cap dv_2 \cap .... \cap dv_{n-2}, Y_{n-3})(dv_{n-1}, X_{n-1}) \ \wedge \ \Psi(Y_{n-2})) \ \wedge && \textit{- Step n-2} \\
&\quad (\exists Y_{n-1} \in U(dv_1 \cap dv_2 \cap .... \cap dv_{n-1}, Y_{n-2})(dv_n, X_n) \ \wedge \ \Psi(Y_{n-1}))) && \textit{- Step n-1}
\end{aligned}$$

Thus, each step in the algorithm considers a unification set using the binary unification function U. The ith 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 $i+1$st step. A depiction of Π_U, with $n = 4$, is given in figure 4. It should be apparent that consistency checking is implicit in each step. Thus, the existence of an internally valid ith 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.

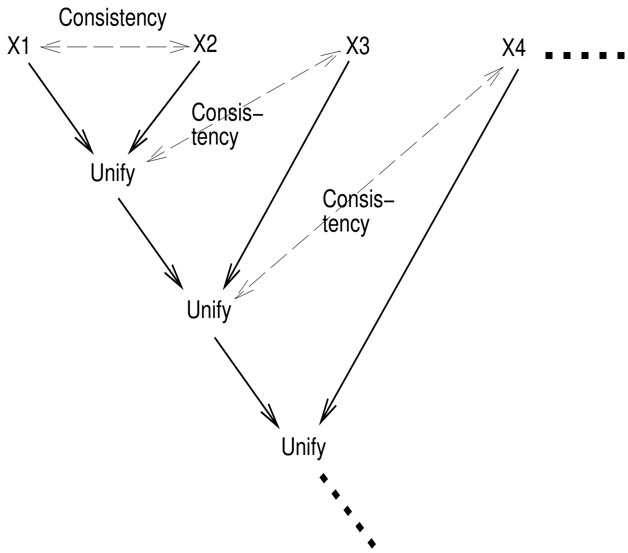

Figure 3: Binary Consistency Algorithm

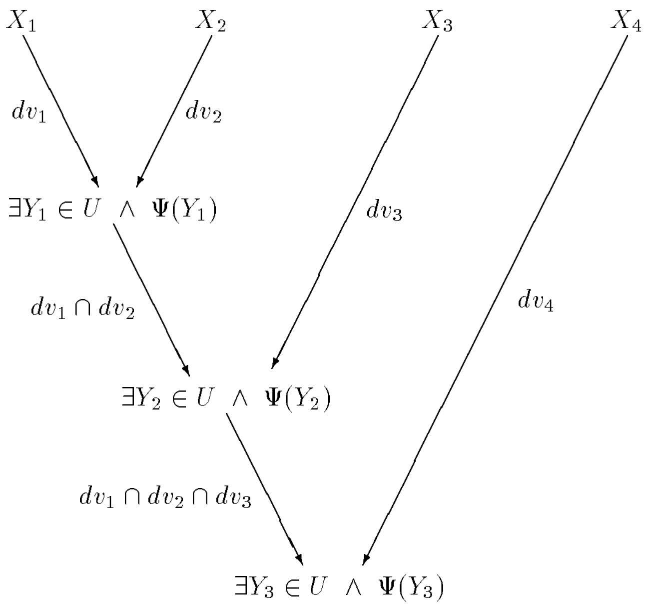

Figure 4: Formal Depiction of Binary Consistency Algorithm

In addition notice how when we have performed a binary unification, e.g. $U(dv, Y)(dv_i, X_i)$, 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_i 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 14
A binary unification function U is valid if and only if,

(U.i) $U(dv, X)(dv', X') \subseteq \mathcal{U}(dv, X)(dv', X')$ *and*
(U.ii) $\mathcal{U}(dv, X)(dv', X') \neq \emptyset \implies U(dv, X)(dv', X') \neq \emptyset$.

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 $\mathcal{U}$ (remember $\mathcal{U}$ is our base unification function, see definiton 4) 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 dv_i, $1 \leq i \leq n$, is a preorder and U satisfies (U.i) and (U.ii),
$\overset{1..n}{\Pi_U}(dv_i, X_i) \implies \overset{1..n}{C}(dv_i, X_i)$.

Proof
Assume $\overset{1..n}{\Pi_U}(dv_i, X_i)$ holds. Now from step n-1 in Π_U we deduce:

$$\begin{aligned} &\exists Y \ s.t.\ Y \ (dv_1 \cap dv_2 \cap ... \cap dv_{n-1}) \ Y_{n-2} \ \wedge && (1)\\ &Y \ dv_n \ X_n \ \wedge && (2)\\ &\Psi(Y) && (3) \end{aligned}$$

We will show that Y is the required common development of X_1 through to X_n to give us $\overset{1..n}{C}(dv_i, X_i)$. Firstly, (2) and (3) give us immediately that $\Psi(Y)$ and $Y \ dv \ X_n$. Now from (1) and $Y_{n-2} \in U(dv_1 \cap ... \cap dv_{n-2}, Y_{n-3})(dv_{n-1}, X_{n-1})$ we can deduce that $Y \ dv_{n-1} \ Y_{n-2}$ and $Y_{n-2} \ dv_{n-1} \ X_{n-1}$, thus, from transitivity of dv_{n-1} we have $Y \ dv_{n-1} \ X_{n-1}$. We can perform similar arguments down through the construction of Π to determine that $Y \ dv_{n-2} \ X_{n-2} \ \wedge ... \wedge$ $Y \ dv_2 \ X_2 \ \wedge \ Y \ dv_1 \ X_1$. Thus, Y is the required common development and $\overset{1..n}{C}(dv_i, X_i)$ holds. □

Using this result we can show that performing Π 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 Π.

Proposition 7
Assume dv_i, $1 \leq i \leq n$, is a preorder. Then,

$$\Pi_{\mathcal{U}}(dv_1, X_1)(dv_2, X_2)...(dv_n, X_n) = \overset{1..n}{C}(dv_i, X_i).$$

Proof
($\Longrightarrow$) $\mathcal{U}$ trivially satisfies (U.i) and (U.ii); thus we can use the previous result, proposition 6, to give this direction of implication.
($\Longleftarrow$) Assume $\overset{1..n}{C}(dv_i, X_i)$ holds, i.e. $\exists Y \in \overset{1..n}{\mathcal{U}}(dv_i, X_i)$ such that $\Psi(Y)$. We will show that Y can act as the unification in all steps of Π. Firstly, the internal validity requirement of each step will clearly be satisfied for Y. In addition, using corollary 1 we get, $Y \in \mathcal{U}(dv_1, X_1)(dv_2, X_2)$ and thus step 1. Step 2 follows since $Y \ dv_3 \ Y_3$, by our assumption and $Y \ dv_1 \cap dv_2 \ Y$ from the reflexivity of development, i.e. $Y \in \mathcal{U}(dv_1 \cap dv_2, Y)(dv_3, X_3)$. Using similar arguments we can get step 3 and all steps up to n-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. $C \not\Longrightarrow \Pi_U$, and we clearly require this direction if Π is to be used.

Example 1 *We will give two simple 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,* $P_1 := i; a; stop[]i; b; stop$, $P_2 := a; stop[]i; b; stop$ *and* $P_3 := a; stop$. *Further consider the consistency check* $C_{\mathbf{red}}(P_1, P_2, P_3)$, *where* **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* $P := i; b; stop$, *and* $C_{\mathbf{red}}(P, P_3)$ *does not hold.*

Z. *Consider the three Z specifications,* $S_1 = [n! : \mathbb{N} \mid n! = 5 \ \vee \ n! = 7]$, $S_2 = [n! : \mathbb{N} \mid n! = 5 \ \vee \ n! = 7]$, *and* $S_3 = [n! : \mathbb{N} \mid n! = 5]$. *The first two specifications could be unified to yield* $[n! : \mathbb{N} \mid n! = 7]$, *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 consider the issue of least developed unifications next.

4.3 Least Developed Unifications

In traditional single threaded (waterfall) models of system development the issue of 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 least developed unification. We assume dv_i, $1 \leq i \leq n$, are preorders.

Definition 15 (Least Developed Unification)
$X \in \overset{1..n}{\mathcal{U}}(dv_i, X_i)$ *is a least developed unification iff* $\neg(\exists X' \in \overset{1..n}{\mathcal{U}}(dv_i, X_i),\ s.t.\ X\ \overline{\overset{n}{\cap} dv_i}\ X')$,
where $\overset{n}{\cap} dv_i$ *is a shorthand for* $dv_1 \cap \ldots \cap dv_n$.

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 $dv_1 \cap \ldots \cap dv_n$, i.e. the set of unifications is ordered by the intersection of the development relations used in unification. Figure 5 depicts a typical situation, X, X' and X'' are unifications of X_1 and X_2 and X, X' and X'' are ordered by $dv_1 \cap dv_2$. In this diagram X is the least developed unification of X_1 and X_2. $dv_1 \cap \ldots \cap dv_n$ is a natural interpretation of development between unifications because all descriptions in the unification set that are descendents of a least developed unification X are developments of X by all relevant development relations.

Note that another way of looking at the least development is that it is a maximal element in the set of possible developments. Thus, by reversing the point of reference we can exchange least 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 least developed of the set of unifications is a problematic concept. Specifically, descriptions in the unification set, $\overset{1..n}{\mathcal{U}}(dv_i, X_i)$, 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 $dv_1 \cap .. \cap dv_n$. Thus, this definition and the remaining theory will only be applied to intra language consistency. Ongoing work is addressing generalisation of least 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 least 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 $\asymp_{\overset{n}{\cap} dv_i}$ 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.)

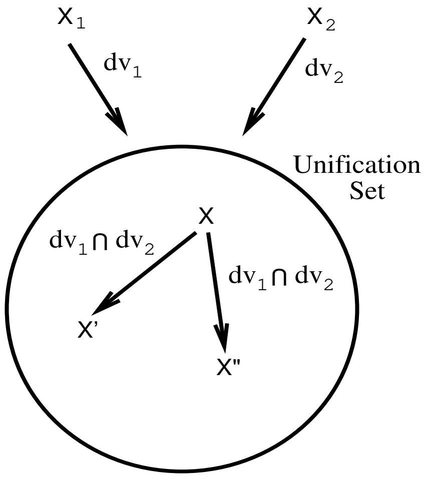

Figure 5: A Typical Least Developed Unification Situation

Example 2 *If we have four descriptions;* X_1, X_2, X_3 *and* X_4*; and the development relations between descriptions indicated in figure 6, both* X_3 *and* X_4 *are least developed unifications of* X_1 *and* X_2*, i.e. they are clearly both in* $\mathcal{U}(dv_1, X_1)(dv_2, X_2)$ *and neither has an ancestor by* $dv_1 \cap dv_2$ *in* $\mathcal{U}(dv_1, X_1)(dv_2, X_2)$*. Furthermore, examples of this form are characteristic of situations that foil* Π*. Specifically, consider the development relations in figure 7. In this situation we may unify* X_1 *and* X_2 *to* X_4 *and then 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 $C(\leq_{tr}, X_1)(\mathbf{ext}, X_2)(\leq_{tr}, X_3)$*, where,*

$$X_1 := a; b; c; stop \quad X_2 := a; stop \quad X_3 := a; stop[]c; stop \quad and \quad X_4 := a; b; stop$$

Now both X_2 *and* X_4 *(amongst others) are least unifications from within* $\mathcal{U}(\leq_{tr}, X_1)(\mathbf{ext}, X_2)$ *(a subset of the relevant development relations is shown in figure 8). In particular, notice that* X_2 *and* X_4 *are not related by* $\leq_{tr} \cap \mathbf{ext}$*, thus, neither is less developed than the other. Furthermore, if we now consider the full consistency check,* $C(\leq_{tr}, X_1)(\mathbf{ext}, X_2)(\leq_{tr}, X_3)$*, the choice of least 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 least 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 least developed unification into two parts. First, we will consider the situation in which the least 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 least 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 least 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 least developed unification set and locate under what circumstances we can take just one element from the set.

5 Non Unique Least Developed Unification

5.1 Relating Consistency and Least Development

We define the least developed unification set, which we denote $\mathcal{LU}(dv_1, X_1)....(dv_n, X_n)$ or $\overset{1..n}{\mathcal{LU}}(dv_i, X_i)$, as follows:-

Definition 16 (Least Developed Unification Set)

$\overset{1..n}{\mathcal{LU}}(dv_i, X_i) = \ \{X : X \in \overset{1..n}{\mathcal{U}}(dv_i, X_i) \ \wedge \ \neg(\exists X' \in \overset{1..n}{\mathcal{U}}(dv_i, X_i), \ s.t. \ X \ \overline{\overset{n}{\cap} dv_i} \ X'\}.$

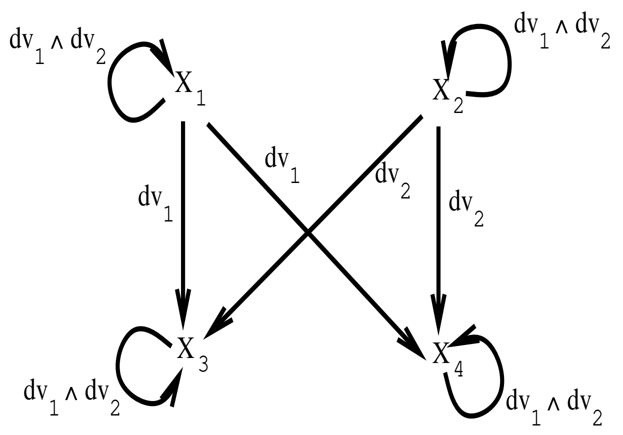

Figure 6: Development Relations

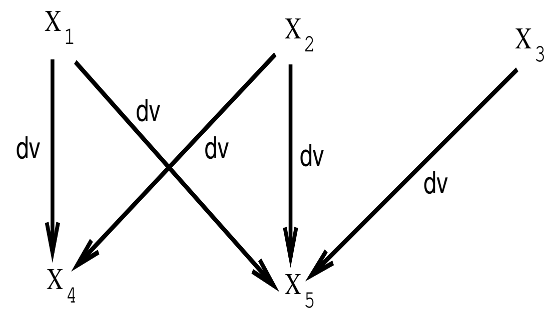

Figure 7: More Development Relations

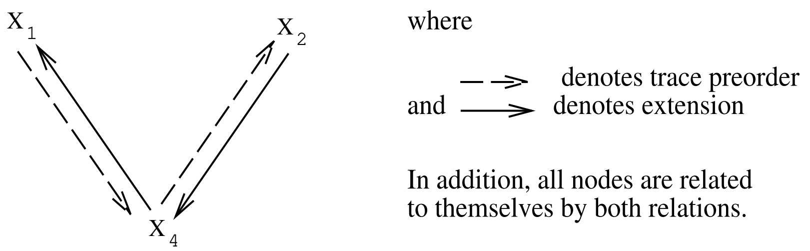

Figure 8: LOTOS Development Relations

Thus, the least 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 $\mathcal{LU}$ as the unification function in Π we must show that $\mathcal{LU}$ 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
$\overset{1..n}{\mathcal{LU}}(dv_i, X_i) \subseteq \overset{1..n}{\mathcal{U}}(dv_i, X_i)$.

Proof
Take $X \in \overset{1..n}{\mathcal{LU}}(dv_i, X_i)$, by the definition of $\mathcal{LU}$, $X \in \overset{1..n}{\mathcal{U}}(dv_i, X_i)$. □

Corollary 2
$\mathcal{LU}(dv, X)(dv', X') \subseteq \mathcal{U}(dv, X)(dv', X')$

(U.ii) though is more difficult and obtaining this validity constraint is central to showing that $\Pi_{\mathcal{LU}}$ 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. 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
$X \in \overset{1..n}{\mathcal{U}}(dv_i, X_i) \implies \exists X' \in \overset{1..n}{\mathcal{LU}}(dv_i, X_i)$ *s.t.* $X \overset{n}{\cap} dv_i \; X'$.

This property states that all unifications have an ancestor in the least developed unification set. In other words, all unifications are developments, by $\overset{n}{\cap} dv_i$, of a least developed unification. Notice, a least developed unification is a development of itself. Further notice, implicit in the condition of the unification $\overset{1..n}{\mathcal{U}}(dv_i, X_i) \neq \emptyset$. 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 corollary of the following more general result:-

Proposition 9
Property 1 $\implies (\overset{1..n}{\mathcal{U}}(dv_i, X_i) \neq \emptyset \implies \overset{1..n}{\mathcal{LU}}(dv_i, X_i) \neq \emptyset)$.

Proof
Assume $\overset{1..n}{\mathcal{U}}(dv_i, X_i) \neq \emptyset$ and take $X \in \overset{1..n}{\mathcal{U}}(dv_i, X_i)$. Now we can use property 1 to get $\exists X' \in \overset{1..n}{\mathcal{LU}}(dv_i, X_i)$. So, $\overset{1..n}{\mathcal{LU}}(dv_i, X_i) \neq \emptyset$, as required. □

Corollary 3
Property 1 $\implies (\mathcal{U}(dv, X)(dv', X') \neq \emptyset \implies \mathcal{LU}(dv, X)(dv', X') \neq \emptyset)$.

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 $\Pi_{\mathcal{LU}}$.

Theorem 1
Given property 1 and $\mathcal{LU}$ a representative unification strategy,

$$\overset{1..n}{C}(dv_i, X_i) = \Pi_{\mathcal{LU}}(dv_1, X_1)...(dv_n, X_n)$$

Proof
The first section of the appendix contains an induction proof (proposition 21), where $\overline{\Pi}_{\mathcal{LU}}$ is a slightly stronger constraint than $\Pi_{\mathcal{LU}}$, that $\overset{1..n}{C}(dv_i, X_i) \implies \overline{\Pi}_{\mathcal{LU}}(dv_1, X_1)...(dv_n, X_n)$. From an examination of the conditions of $\overline{\Pi}$, if $\mathcal{LU}$ is representative, $\overline{\Pi}_{\mathcal{LU}}(dv_1, X_1)...(dv_n, X_n) \implies \Pi_{\mathcal{LU}}(dv_1, X_1)...(dv_n, X_n)$. In addition, proposition 6 gives us the other direction of implication. □

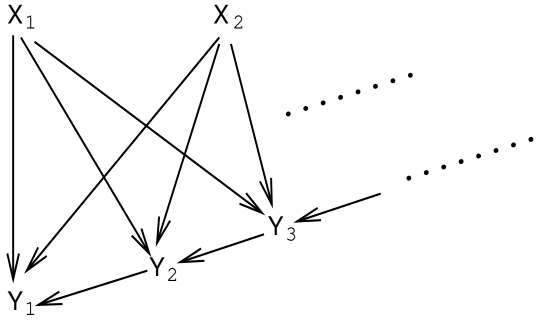

Figure 9: Infinite chain of candidate 'least' developed unifications

This is the result we are seeking, it states that subject to property 1 holding and $\mathcal{LU}$ a representative unification strategy we can equate the binary consistency checking strategy $\Pi_{\mathcal{LU}}$ with consistency, i.e. if each binary unification considers the set of all least developed unifications then we will obtain consistency. However, in order, to obtain this identity we 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 least unifications may be infinite, as depicted in figure 9 and a maximal member of the chain, Y_i, may not exist. This is unlikely to arise in practice, but, is theoretically possible for arbitrary preorders. Notice that it is certain that the unification set can be infinite, e.g. consider the LOTOS **ext** relation. We would like to locate a constraint on development that prevents the unification set being infinitely increasing in the manner highlighted. Property 1 states:

$$X \in \overset{1..n}{\mathcal{U}}(dv_i, X_i) \implies \exists X' \in \overset{1..n}{\mathcal{LU}}(dv_i, X_i) \ s.t. \ X \overset{n}{\cap} dv_i \ X'.$$

i.e. if the unification set is non-empty all unifications are descendents of a least developed unification. In order to characterise when this property can be obtained we need some definitions.

Definition 17 *For $S \subseteq DES$ and $dv \in DEV$,*

$$M(S, dv) = (\exists Y \in S \ s.t. \ \neg(\exists Y' \in S \ s.t. \ Y \ \overline{dv} \ Y')).$$

Such a Y is called a maximal element of S.

Thus, $M(S, dv)$ will hold if and only if the set S of descriptions has a maximal element by dv, i.e. an element, Y, which has no ancestor by dv in S. When we are considering maximal elements of unification sets we will talk about *maximal unifications.*

The next two definitions are interpretations of standard mathematical concepts, see for example [13].

Definition 18 *An infinite set of descriptions $\{X_1, X_2, ...\}$ is said to be an infinitely ascending chain $X_1, X_2, ...$ according to dv iff $X_i \ \overline{dv} \ X_{i+1}$ for all $i \in \mathbb{N}$.*

Definition 19 (Well Founded Set)
S is called a well founded (WF) set by dv iff $\forall S' \subseteq S$, $(S' \neq \emptyset \implies M(S', dv))$.

Thus, a partial order (S, dv) is well founded (WF) if and only if all non-empty subsets of S have at least one maximal element. Clearly, we could consider dual definitions which focus on the opposite direction of the development partial order, e.g. minimal elements of ancestors by development. However, our focus is on evolution of descriptions towards development.

Notice that a maximal element of a set is not necessarily unique up to equivalence. There could be a number of unifications with no ancestor by development in the unification set, see for example figure 6.

The following is a standard result from mathematical set theory, see [13] for example (a proof of the result is reproduced in [4]).

Proposition 10
(i) (S, dv) *is well founded.*

$$\Longleftrightarrow$$

(ii) There is no infinitely ascending chain in (S, dv).

With these concepts we can characterise under what circumstances property 1 can be obtained.

Proposition 11
(i) There is no infinite ascending chain by $\overline{\overset{n}{\sqcap} dv_i}$ *of descriptions in* $\overset{1..n}{\mathcal{U}}(dv_i, X_i)$.

$$\Longrightarrow$$

(ii) $X \in \overset{1..n}{\mathcal{U}}(dv_i, X_i) \implies \exists X' \in \mathcal{L}\overset{1..n}{\mathcal{U}}(dv_i, X_i)$ *s.t.* $X \overset{n}{\sqcap} dv_i\ X'$.

i.e. (i) $\Longrightarrow$ *property 1.*

Proof
By contradiction, so, assume (i). Now $\neg$(ii) gives:
$\exists X \in \overset{1..n}{\mathcal{U}}(dv_i, X_i)$ *s.t.* $\neg(\exists X' \in \mathcal{L}\overset{1..n}{\mathcal{U}}(dv_i, X_i)$ *s.t.* $X \overset{n}{\sqcap} dv_i\ X')$.
Now consider the following construction:-

1. $X_0 = X$.
2. Select $X_1 \in \overset{1..n}{\mathcal{U}}(dv_i, X_i)$ such that $X_0\ \overline{\overset{n}{\sqcap} dv_i}\ X_1$. Such an X_1 must exist, otherwise X_0 would be a least developed unification and a development by $\overset{n}{\sqcap} dv_i$ of itself, which contradicts out assumption of $\neg$(ii).
3. If $X_0, X_1, ..., X_j \in \overset{1..n}{\mathcal{U}}(dv_i, X_i)$ for $j \geq 0$, such that $X_0\ \overline{\overset{n}{\sqcap} dv_i}\ X_1 \ \wedge\ X_1\ \overline{\overset{n}{\sqcap} dv_i}\ X_2 \ \wedge\ ...\ \wedge\ X_{j-1}\ \overline{\overset{n}{\sqcap} dv_i}\ X_j$, have already been chosen, then a description X_{j+1} such that $X_j\ \overline{\overset{n}{\sqcap} dv_i}\ X_{j+1}$ can be found. Such an X_{j+1} must exist otherwise X_j would be a least developed unification and by transitivity of development and an ancestor by $\overline{\overset{n}{\sqcap} dv_i}$ of X_0, which would contradict our assumption of $\neg$(ii).

This construction will generate an infinite ascending chain by $\overline{\overset{n}{\sqcap} dv_i}$ of descriptions $X_0, X_1, X_2, ... \in \overset{1..n}{\mathcal{U}}(dv_i, X_i)$, which contradicts our assumption of (i) as required. □

Using this result we can obtain the following important corollary:-

Corollary 4
(i) $(\overset{1..n}{\mathcal{U}}(dv_i, X_i), \overset{n}{\sqcap} dv_i)$ *is well founded.*

$$\Longrightarrow$$

(ii) Property 1.

This result characterises the properties that are required of $\overset{1..n}{\mathcal{U}}(dv_i, X_i)$ 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 up holds the following:

Property 2
FDT ft satisfies property 2 iff

$$\forall X_1, ..., X_n \in DES_{ft} \ \wedge\ \forall dv_1, ..., dv_n \in DEV_{ft}\ (\overset{1..n}{\mathcal{U}}(dv_i, X_i), \overset{n}{\sqcap} dv_i) \textit{ is well founded}$$

Another way to express property 2 is:

$$\forall X_1, ..., X_n \in DES_{ft} \ \wedge \ \forall dv_1, ..., dv_n \in DEV_{ft}, \ (\overset{n}{\sqcap} D(x_i, dv_i), \overset{n}{\sqcap} dv_i) \textit{ is well founded.}$$

since $\overset{n}{\sqcap} D(x_i, dv_i) = \overset{1..n}{\mathcal{U}}(dv_i, X_i)$.

In order to verify that property 2 holds for a particular FDT we would like to obtain a constraint that can be realistically checked for actual development relations. Thus, we consider a series of "Well Behavedness" properties on development. These results are listed in the appendix, but not proved. The interested reader is referred to [4] for a complete discussion of these properties and proofs of results. If a particular FDT can be shown to satisfy any of these properties then property 2 and hence property 1 will follow, which in turn imply that a binary consistency checking strategy using least developed unification sets can be safely used.

6 Unique Least 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 least developed unifications is likely to be significantly smaller than the full unification set, it could still be very large. This subsection considers under what circumstances we can safely select any member from the set of least 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 20 *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 $g(S, dv) = \perp$.*

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 their 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 $g(\overset{1..n}{\mathcal{U}}(dv_i, X_i), \overset{n}{\sqcap} dv_i)$ the greatest unification.

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

Proposition 12
(i) A greatest element is a maximal element.
(ii) If it exists, $g(\overset{1..n}{\mathcal{U}}(dv_i, X_i), \overset{n}{\sqcap} dv_i) \in \overset{1..n}{\mathcal{LU}}(dv_i, X_i)$, i.e. the greatest element is a least developed unification.
(iii) A greatest element is unique up to equivalence.
(iv) Assuming property 1 $\forall X, X' \in \overset{1..n}{\mathcal{LU}}(dv_i, X_i), \ X \asymp_{\overset{n}{\sqcap} dv_i} X' \iff g(\overset{1..n}{\mathcal{U}}(dv_i, X_i), \overset{n}{\sqcap} dv_i) \neq \perp$.

The last of these results is important as it shows that the existence of a greatest unification is the only circumstance that will yield a unique least developed unification, i.e. the least 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 least developed unifications, is that it has a greatest element, i.e.

Property 3
If $\overset{1..n}{\mathcal{U}}(dv_i, X_i) \neq \emptyset$ then $g(\overset{1..n}{\mathcal{U}}(dv_i, X_i), \overset{n}{\sqcap} dv_i) \neq \perp$.

We assume the following greatest unification function, $\mathcal{L}$,:

Definition 21
If $g(\overset{1..n}{\mathcal{U}}(dv_i, X_i), \overset{n}{\sqcap} dv_i) = \perp$ then $\overset{1..n}{\mathcal{L}}(dv_i, X_i) = \emptyset$ otherwise $\overset{1..n}{\mathcal{L}}(dv_i, X_i) = \{g(\overset{1..n}{\mathcal{U}}(dv_i, X_i), \overset{n}{\sqcap} dv_i)\}$.

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 $\mathcal{L}$ up holds (U.i) and (U.ii). These arise as immediate consequences of the next results (proofs can be found in [4].

Proposition 13
Given property 3,

(i) $\overset{1..n}{\mathcal{L}}(dv_i, X_i) \subseteq \overset{1..n}{\mathcal{U}}(dv_i, X_i)$.
(ii) $\overset{1..n}{\mathcal{U}}(dv_i, X_i) \neq \emptyset \implies \overset{1..n}{\mathcal{L}}(dv_i, X_i) \neq \emptyset$

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

Property 4
$X \in \overset{1..n}{\mathcal{U}}(dv_i, X_i) \implies X \overset{n}{\cap} dv_i\ Y$ *where* $Y \in \overset{1..n}{\mathcal{L}}(dv_i, X_i)$.

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

Proposition 14
Property 3 $\implies$ *property 4.*

Proof
$\overset{1..n}{\mathcal{U}}(dv_i, X_i) \neq \emptyset \implies \overset{1..n}{\mathcal{L}}(dv_i, X_i) \neq \emptyset$, the result follows immediately from the definition of $\mathcal{L}$. □

We will also use the following simple result

Proposition 15
Given property 3,
$Y \in \mathcal{L}(dv, X)(dv', X') \ \wedge\ Y' \in \mathcal{L}(dv, X)(dv', X')(dv'', X'') \implies Y'\ dv \cap dv'\ Y$.

Proof
Clearly, $Y' \in \mathcal{U}(dv, X)(dv', X')(dv'', X'')$, but we can use corollary 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 β 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 22
$\beta(dv, X)(dv', X') = (dv \cap dv', Y)$
  where
    if $X = \perp\ \vee\ X' = \perp\ \vee\ \mathcal{L}(dv, X)(dv', X') = \emptyset$ *then* $Y = \perp$
    otherwise $Y \in \mathcal{L}(dv, X)(dv', X')$.

We will prove associativity of β by relating the two possible binary bracketings of β to $\mathcal{L}(dv, X)(dv', X')(dv'', X'')$.

Proposition 16
Given property 3,
$r(\beta(dv, X)(\beta(dv', X')(dv'', X''))) \asymp_{dv \cap dv' \cap dv''} Y$ *where* $Y \in \mathcal{L}(dv, X)(dv', X')(dv'', X'')$ *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 $Y' \in \mathcal{L}(dv, X)(dv', X')(dv'', X'')$. By transitivity of development $Y \in \mathcal{U}(dv, X)(dv', X')(dv'', X'')$, so by the definition of $\mathcal{L}$ we get $Y\ dv \cap dv' \cap dv''\ Y'$. Also, let $Y'' = r(\beta(dv', X')(dv'', X''))$. By, proposition 15 $Y'\ dv' \cap dv''\ Y''$. 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, $Y \in \mathcal{L}(dv, X)(dv' \cap dv'', Y'')$, 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, $Y \asymp_{dv \cap dv' \cap dv''} Y'$, as required. □

Proposition 17
Given property 3,
$r(\beta(\beta(dv,X)(dv',X'))(dv'',X'')) \asymp_{dv \cap dv' \cap dv''} Y$ *where* $Y \in \mathcal{L}(dv,X)(dv',X')(dv'',X'')$

Proof
Similar to proof of proposition 16. □

Now if we define equality pairwise as,

$$(dv,X) = (dv',X') \text{ iff } dv = dv' \ \wedge \ X \asymp_{dv \cap dv'} X'$$

the following result is straightforward.

Corollary 5 *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 16 and 17. □

This is a full associativity result which gives us that any bracketing of $\beta(dv_1,X_1),...,(dv_n,X_n)$ is equal. Since β is just an alternative coding of $\mathcal{L}$ that facilitates clarity of expression, we have full associativity of $\mathcal{L}$ and that a consistency strategy using $\mathcal{L}$ can be composed of any ordering of binary consistency checks, in particular, $\Pi_{\mathcal{L}} = C$. 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 least developed unification set. What conditions can we impose on development in order to obtain the existence of such a greatest element?

In a similar way to 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,*

$$\forall X_1,...,X_n \in DES_{ft} \ \wedge \ \forall dv_1,...,dv_n \in DEV_{ft}, \ (\overset{1..n}{\mathcal{U}}(dv_i,X_i) \neq \emptyset \implies g(\overset{1..n}{\mathcal{U}}(dv_i,X_i), \overset{n}{\cap} dv_i) \neq \perp).$$

This property ensures that any possible combination of descriptions and development relations in ft 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 least developed unification set. The interested reader is referred to the appendix for a list of well behavedness properties on development which imply this property.

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 this paper 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 23 $X \in DES_{ft}$ *is a lower bound of* $Z \subseteq DES_{ft}$ *iff* $\forall X' \in Z,\ X \ dv \ X'$. *The set of all lower bounds of* Z *is denoted,* $lb(Z,dv)$. *If a lower bound does not exist* $lb(Z,dv) = \emptyset$

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 maximal or greatest element. It should be clear that for balanced consistency lower bounds correspond to unifications, i.e. $\mathcal{U}_{dv}(X_1,...,X_n) = lb(\{X_1,...,X_n\},dv)$. 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 lb.

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

Definition 24 *For $Z \subseteq DES_{ft}$ $glb(Z, dv)$ is a lower bound such that all other lower bounds are a development of $glb(Z, dv)$; i.e. $glb(Z, dv) \in lb(Z, dv) \ \wedge \ (\forall X \in lb(Z, dv), \ X \ dv \ glb(Z, dv))$. If a greatest lower bound does not exist $glb(Z, dv) = \perp$.*

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 $\overset{n}{\sqcap} dv_i$ in the general (unbalanced) case has been collapsed to just dv.

We can now define consistency in this restricted setting:-

Definition 25 $C_{dv}(X_1, ..., X_n) \iff \exists X \in lb(\{X_1, ..., X_n\}, dv)$ *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 18
$\forall Z \subseteq DES_{ft} \ \wedge \ dv \in DEV_{ft}, \ \exists X \in lb(Z, dv) \ \wedge \ \Psi(X) \iff \forall X_1, ..., X_n \in DES_{ft}, \ C_{dv}(X_1, ..., X_n)$ *holds.*

Proof
Straightforward. □

i.e. if all subsets of DES_{ft} 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 $\forall X \in DES_{ft}, \ \perp_{dv} \ dv \ X$.

Proposition 19
DES_{ft} has an internally valid bottom element $\Longrightarrow \forall X_1, .., X_n \in DES_{ft}, \ C_{dv}(X_1, ..., X_n)$ *holds.*

Proof
Immediate. □

Example 3 *As a simple illustration for LOTOS $C_{\leq_{tr}}$ and $C_{\mathbf{ext}}$ are completely consistent, since all groups of specifications have common refinements. For example, the process stop, which offers only the empty trace is a bottom element for $\leq_{tr}$ and the process that offers a choice of all possible actions at all points in the computation is a bottom 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 20
$glb(\{X_1, ..., X_n\}, dv) \neq \perp \implies glb(\{X_1, ..., X_n\}, dv) \in \mathcal{L}_{dv}(X_1, ..., X_n)$.

Proof
By definition. □

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

Property 6
$\forall \{X_1, ..., X_n\} \subseteq DES_{ft} \ \wedge \ \forall dv \in DEV_{ft}, \ lb(\{X_1, ..., X_n\}, dv) \neq \emptyset \implies glb(\{X_1, ..., X_n\}, dv) \neq \perp$.

This property ensures that if a lower bound exists then a greatest lower bound can be found, i.e. the unification of $X_1, ..., X_n$ 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 26 *(DES_{ft}, dv) is cocomplete iff $\forall S \subseteq DES_{ft}, \ glb(S, dv) \neq \perp$.*

Cocompleteness is related to the standard concept of a complete partial order, see for example [13], 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.

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 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 identifed. These range from the very poorly behaved to the very well behaved. These classes are summarised in the following table.

Class of Consistency			**Implications**
Unbalanced	Inter lang.		No results
Unbalanced	Intra lang.	Not WF unif. set	No incremental cons. checking
Unbalanced	Intra lang.	WF unif. set	Set of least developed unifications
Unbalanced	Intra lang.	Greatest unifs.	Unique incremental cons. checking
Balanced	Intra lang.	Not WF unif. set	No incremental cons. checking
Balanced	Intra lang.	WF without glb's	Set of least developed unifications
Balanced	Intra lang.	glbs always exist	Unique incremental cons. checking
Balanced	Intra lang.	Cocomplete	Completely consistent and unique incremental cons. checking

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

References

[1] M. Ainsworth, A. H. Cruickshank, L. J. Groves, and P. J. L. Wallis. Viewpoint specification and Z. *Information and Software Technology*, 36(1):43–51, February 1994.

[2] E. Boiten, H. Bowman, J. Derrick, and M. Steen. Cross viewpoint consistency in Open Distributed Processing (intra language consistency). Technical Report 8-95, Computing Laboratory, University of Kent at Canterbury, 1995.

[3] T. Bolognesi and E. Brinksma. Introduction to the ISO Specification Language LOTOS. *Computer Networks and ISDN Systems*, 14(1):25–29, 1988.

[4] H. Bowman, E.A. Boiten, J. Derrick, and M.W.A. Steen. Strategies for consistency checking. Technical Report 12-95, Computing Laboratory, University of Kent at Canterbury, 1995.

[5] H. Bowman, J. Derrick, and M. Steen. Some results on cross viewpoint consistency checking. In K. Raymond and L. Armstrong, editors, *IFIP TC6 International Conference on Open Distributed Processing*, pages 399–412, Brisbane, Australia, February 1995. Chapman and Hall.

[6] E. Brinksma and G. Scollo. Formal notions of implementation and conformance in LOTOS. Technical Report INF-86-13, Dept of Informatics, Twente University of Technology, 1986.

[7] G. Cowen, J. Derrick, M. Gill, G. Girling (editor), A. Herbert, P. F. Linington, D. Rayner, F. Schulz, and R. Soley. *Prost Report of the Study on Testing for Open Distributed Processing.* APM Ltd, 1993.

[8] J. Derrick, H. Bowman, and M. Steen. Maintaining cross viewpoint consistency using Z. In K. Raymond and L. Armstrong, editors, *IFIP TC6 International Conference on Open Distributed Processing*, pages 413–424, Brisbane, Australia, February 1995. Chapman and Hall.

[9] J. Derrick, H. Bowman, and M. Steen. Viewpoints and Objects. In *Ninth Annual Z User Workshop*, Limerick, September 1995. Springer-Verlag. To appear.

[10] S. Easterbrook, A. Finkelstein, J. Kramer, and B. Nuseibeh. Coordinating distributed ViewPoints: The anatomy of a consistency check. *Concurrent Engineering: Research and Applications*, 2(3), 1994.

[11] J. Fiadeiro and T. Maibaum. Temporal theories as modularisation units for concurrent system specification. *Formal Aspects of Computing*, 4:239–272, 1992.

[12] G. Leduc. A framework based on implementation relations for implementing LOTOS specifications. *Computer Networks and ISDN Systems*, 25:23–41, 1992.

[13] J. Loeckx and K. Sieber. *The Foundations of Program Verification.* Wiley, 1984.

[14] B. Potter, J. Sinclair, and D. Till. *An introduction to formal specification and Z.* Prentice Hall, 1991.

[15] I. Sommerville. *Software Engineering.* Addison-Wesley, 1989.

[16] M. Steen, H. Bowman, and J. Derrick. Composition of LOTOS specifications. In P. Dembinski and M. Sredniawa, editors, *Protocol Specification, Testing and Verification*, Warsaw, Poland, 1995. Chapman & Hall.

[17] P. Zave and M. Jackson. Conjunction as composition. *ACM Transactions on Software Engineering and Methodology*, 2(4):379–411, October 1993.

Appendix

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

Results for Section 5

Proposition 21
Given property 1,
$\exists X \in \mathcal{U}(dv_1, X_1)...(dv_m, X_m) \implies \overline{\Pi}_{\mathcal{LU}}(dv_1, X_1)...(dv_m, X_m)$
where
$\overline{\Pi}_{\mathcal{LU}}(dv_1, X_1)...(dv_m, X_m) \; =$
$((\exists Y_1 \in \mathcal{LU}(dv_1, X_1)(dv_2, X_2) \;\wedge\; X \in \mathcal{U}(dv_1, X_1)(dv_2, X_2)) \;\wedge$
$(\exists Y_2 \in \mathcal{LU}(dv_1 \cap dv_2, Y_1)(dv_3, X_3) \;\wedge\; X \in \mathcal{U}(dv_1 \cap dv_2, Y_1)(dv_3, X_3)) \;\wedge$
$(\exists Y_3 \in \mathcal{LU}(dv_1 \cap dv_2 \cap dv_3, Y_2)(dv_4, X_4) \;\wedge\; X \in \mathcal{U}(dv_1 \cap dv_2 \cap dv_3, Y_2)(dv_4, X_4)) \;\wedge$
$\cdots$
$\cdots$
$(\exists Y_{m-2} \in \mathcal{LU}(dv_1 \cap .... \cap dv_{m-2}, Y_{m-3})(dv_{m-1}, X_{m-1}) \;\wedge\; X \in \mathcal{U}(dv_1 \cap .... \cap dv_{m-2}, Y_{m-3})(dv_{m-1}, X_{m-1})) \;\wedge$
$(\exists Y_{m-1} \in \mathcal{LU}(dv_1 \cap .... \cap dv_{m-1}, Y_{m-2})(dv_m, X_m) \;\wedge\; X \in \mathcal{U}(dv_1 \cap .... \cap dv_{m-1}, Y_{m-2})(dv_m, X_m)))$.
Notice that we are not considering Π *directly, rather we consider the unification strategy* $\overline{\Pi}$ *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. We will relate to* Π *as a corollary to this theorem.*

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:

(As) $\exists X \in \mathcal{U}(dv_1, X_1)(dv_2, X_2) \implies$ ((a) $\exists Y_1 \in \mathcal{LU}(dv_1, X_1)(dv_2, X_2) \wedge$
(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:

(As) $\exists X \in \mathcal{U}(dv_1, X_1)(dv_2, X_2)(dv_3, X_3) \implies$
((a) $\exists Y_1 \in \mathcal{LU}(dv_1, X_1)(dv_2, X_2)$ $\wedge$ (b) $X \in \mathcal{U}(dv_1, X_1)(dv_2, X_2) \wedge$
(c) $\exists Y_2 \in \mathcal{LU}(dv_1 \cap dv_2, Y_1)(dv_3, X_3)$ $\wedge$ (d) $X \in \mathcal{U}(dv_1 \cap dv_2, Y_1)(dv_3, X_3)$)

Firstly, by observing that from corollary 1 $X \in \mathcal{U}(dv_1, X_1)(dv_2, X_2)(dv_3, X_3)$ implies that $X \in \mathcal{U}(dv_1, X_1)(dv_2, X_2)$ 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 $\exists Y_1' \in \mathcal{LU}(dv_1, X_1)(dv_2, X_2)$ such that $X\ (dv_1 \cap dv_2)\ Y_1'$ and since $X\ dv_3\ X_3$ from our assumption, (As), we have $X \in \mathcal{U}(dv_1 \cap dv_2, Y_1')(dv_3, X_3)$ which gives us (d) and then we can use (U.ii) to get $\exists Y_2 \in \mathcal{LU}(dv_1 \cap dv_2, Y_1')(dv_3, X_3)$, i.e. (c). This completes the verification of base case 2.

Inductive Step.

We wish to prove that: proposition (1) $\implies$ proposition (2), where,

Proposition (1) states:

(As.i) $\exists X \in \mathcal{U}(dv_1, X_1)...(dv_n, X_n) \implies$
((1.1) $\exists Y_1 \in \mathcal{LU}(dv_1, X_1)(dv_2, X_2)$ $\wedge$ $X \in \mathcal{U}(dv_1, X_1)(dv_2, X_2)$ $\wedge$
(1.2) $\exists Y_2 \in \mathcal{LU}(dv_1 \cap dv_2, Y_1)(dv_3, X_3)$ $\wedge$ $X \in \mathcal{U}(dv_1 \cap dv_2, Y_1)(dv_3, X_3)$ $\wedge$
...
...
...
(1.n-2) $\exists Y_{n-2} \in \mathcal{LU}(dv_1 \cap ... \cap dv_{n-2}, Y_{n-3})(dv_{n-1}, X_{n-1})$ $\wedge$ $X \in \mathcal{U}(dv_1 \cap ... \cap dv_{n-2}, Y_{n-3})(dv_{n-1}, X_{n-1})$ $\wedge$
(1.n-1) $\exists Y_{n-1} \in \mathcal{LU}(dv_1 \cap ... \cap dv_{n-1}, Y_{n-2})(dv_n, X_n)$ $\wedge$ $X \in \mathcal{U}(dv_1 \cap ... \cap dv_{n-1}, Y_{n-2})(dv_n, X_n)$)

Proposition (2) states:

(As.ii) $\exists X \in \mathcal{U}(dv_1, X_1)...(dv_{n+1}, X_{n+1}) \implies$
((2.1) $\exists Y_1 \in \mathcal{LU}(dv_1, X_1)(dv_2, X_2)$ $\wedge$ $X \in \mathcal{U}(dv_1, X_1)(dv_2, X_2)$ $\wedge$
(2.2) $\exists Y_2 \in \mathcal{LU}(dv_1 \cap dv_2, Y_1)(dv_3, X_3)$ $\wedge$ $X \in \mathcal{U}(dv_1 \cap dv_2, Y_1)(dv_3, X_3)$ $\wedge$
...
...
...
(2.n-2) $\exists Y_{n-2} \in \mathcal{LU}(dv_1 \cap ... \cap dv_{n-2}, Y_{n-3})(dv_{n-1}, X_{n-1})$ $\wedge$ $X \in \mathcal{U}(dv_1 \cap ... \cap dv_{n-2}, Y_{n-3})(dv_{n-1}, X_{n-1})$ $\wedge$
(2.n-1) $\exists Y_{n-1} \in \mathcal{LU}(dv_1 \cap ... \cap dv_{n-1}, Y_{n-2})(dv_n, X_n)$ $\wedge$ $X \in \mathcal{U}(dv_1 \cap ... \cap dv_{n-1}, Y_{n-2})(dv_n, X_n)$ $\wedge$
(2.n) $\exists Y_n \in \mathcal{LU}(dv_1 \cap ... \cap dv_n, Y_{n-1})(dv_{n+1}, X_{n+1})$ $\wedge$ $X \in \mathcal{U}(dv_1 \cap ... \cap dv_n, Y_{n-1})(dv_{n+1}, X_{n+1})$)

So, assume proposition (1). It is clear that the first n-1 steps of proposition (2), i.e. (2.1), (2.2), ..., (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, $\exists Y_{n-1} \in \mathcal{LU}(dv_1 \cap ... \cap dv_{n-1}, Y_{n-2})(dv_n, X_n)$ and $X \in \mathcal{U}(dv_1 \cap ... \cap dv_{n-1}, Y_{n-2})(dv_n, X_n)$ from (1.n-1), so we can use property 1 to get that $\exists Y_{n-1}' \in \mathcal{LU}(dv_1 \cap ... \cap dv_{n-1}, Y_{n-2})(dv_n, X_n)$ such that $X\ (dv_1 \cap ... \cap dv_{n-1}) \cap dv_n\ Y_{n-1}'$, which implies that $X \in \mathcal{U}(dv_1 \cap ... \cap dv_n, Y_{n-1}')(dv_{n+1}, X_{n+1})$ since $X\ dv_{n+1}\ X_{n+1}$ from (As.ii). This gives us the second half of (2.n) and the first half follows directly from (U.ii).

By the principle of mathematical induction, the result follows. □

WellBehavedness Properties arising from Section 5.2

The following four well behavedness properties (WBC1, WBC2, WBC3 and WBC4) all imply property 2 (see [4] for proofs). Thus, if any of them can be shown to hold for a particular unification problem then theorem 1 will hold and we will be able to obtain global consistency from binary consistency in the manner highlighted.

The first such well behavedness property states that (i) development sets are well founded and (ii) if two development sets are well founded then the intersection of the development sets is also well founded.

Definition 27 (Well Behaved Condition 1 (WBC1))
For an FDT, ft, we say that development is well behaved (condition 1) iff $\forall X \in DES_{ft} \;\wedge\; \forall dv \in DEV_{ft}$

(i) $(D(X,dv),dv)$ *is WF.*
(ii) $(D(X,dv),dv) \;\wedge\; (D(X',dv'),dv')$ *are WF* $\Longrightarrow$ $(D(X,dv) \cap D(X',dv'), dv \cap dv')$ *is WF.*

A stronger formulation of the second of these conditions, (WBC1.ii), that may be easier to prove is:

$$\forall Z \in DES_{ft},\ (Z,dv) \;\wedge\; (Z',dv') \text{ are WF} \Longrightarrow (Z \cap Z', dv \cap dv') \text{ is WF.}$$

This condition is stronger since it is defined over all subsets of DES_{ft}, not just the subsets that are development sets by dv and dv'. It should also be clear that from associativity of $\wedge$ and $\cap$, (WBC1.ii) implies:

$$(D(X_1,dv_1),dv_1) \;\wedge .... \wedge\; (D(X_n,dv_n),dv_n) \text{ are WF} \Longrightarrow (\overset{n}{\cap} D(X_i,dv_i), \overset{n}{\cap} dv_i) \text{ is WF.}$$

An alternative is the following condition:-

Definition 28 (Well Behaved Condition 2 (WBC2))
Development is well behaved (condition 2) in FDT ft iff $\forall dv_1, ..., dv_n \in DEV_{ft}$ $(DES_{ft}, \overset{n}{\cap} dv_i)$ *is well founded.*

This states that all non-empty subsets of DES_{ft} have a maximal element by $\overset{n}{\cap} dv_i$. This is clearly a strong condition as it acts over all subsets of DES_{ft} not just those arising from development.

Both WBC1 and WBC2 in some way impose well behavedness constraints on $\overset{n}{\cap} dv_i$, i.e. they require that the intersection of the development relations being used are well behaved in some sense. This focus on the intersection of development relations is not ideal. It would be better if we could check a well behavedness property on each of the development relations individually and not have to consider the interplay of these relations when their intersection is taken. In this way we would be able to check all the development relations individually for a particular FDT and know that we can intersect them as we like. An obvious constraint to consider is well foundedness of constituent development relations, i.e. can we deduce that $\overset{n}{\cap} dv_i$ is well founded if dv_i is well founded for all $1 \leq i \leq n$. Unfortunately, this does not turn out to be straightforward see [4]. The closest general result we can get is the following:

Definition 29 (Well Behaved Condition 3 (WBC3))
Development is well behaved (condition 3) in FDT, ft, iff

(i) $\forall dv \in DEV_{ft}, dv$ *is well founded.*

(ii) $\forall dv, dv' \in DEV_{ft},\ \asymp_{dv} = \asymp_{dv'}$.

So, we have failed to push well behavedness totally out to checks on individual development relations, i.e. we still need to relate equivalence in the distinct development relations. However, the following very strong constraint will succeed in this respect. If development yields a finite development set then property 2 follows. In some circumstances this very strong condition will be sufficient to obtain the result we require.

Definition 30 (Well Behaved Condition 4 (WBC4))
For an FDT, ft, we say development is well behaved (condition 4) iff, $\forall X \in DES_{ft} \;\wedge\; \forall dv \in DEV_{ft},\ D(X,dv)$ *is finite.*

Well Behavedness Conditions arising from Section 6

The following two well behavedness conditions imply property 5. They play a similar role to the conditions WBC1, WBC2, WBC3 and WBC4 considered in the previous subsection of this appendix.

The first condition that will ensure property 5 corresponds to WBC1 of section 5.2.

Definition 31 (Well Behaved Condition a (WBCa))
For an FDT, ft, development is well behaved (condition a) iff $\forall X \in DES_{ft} \;\wedge\; \forall dv \in DEV_{ft}$

(i) $g(D(X, dv), dv) \neq\perp$

(ii) $g(D(X, dv), dv) \neq\perp \ \wedge g(D(X', dv'), dv') \neq\perp \ \wedge D(X, dv) \cap D(X', dv') \neq \emptyset \implies g(D(X, dv) \cap D(X', dv'), dv \cap dv') \neq\perp$

The following is an alternative condition that corresponds to WBC2 of section 5.2.

Definition 32 (Well Behaved Condition b (WBCb))
For an FDT, ft, development is well behaved (condition b) iff $\forall dv_1, .., dv_n \in DEV_{ft}, \forall S \subseteq DES_{ft},\ (S \neq \emptyset \implies g(S, \overset{n}{\cap} dv_i) \neq\perp)$.

In a similar way to in the previous section we would also like to derive a property that we can check solely on individual development relations, without having to consider the interplay of these relations on intersection. The following proposition demonstrates that this cannot be easily obtained.

Proposition 22
$\forall S \subseteq DES_{ft}$ *s.t.* $S \neq \emptyset$,

(i) $g(S, dv) \neq\perp$ *and* $g(S, dv') \neq\perp$

$\not\Longrightarrow$

(ii) $g(S, dv \cap dv') \neq\perp$.

Proof
By counterexample. So, assume (i), i.e. $\forall S \subseteq DES_{ft}$ *s.t.* $S \neq \emptyset$, $g(S, dv) \neq\perp$ and $g(S, dv') \neq\perp$. Consider the set $S' \subseteq DES_{ft}$ with two elements, i.e. $S' = \{X_1, X_2\}$ and assume that $X_1\ dv\ X_2$, $X_2\ dv'\ X_1$ and the identities hold (i,e. $X_1\ dv\ X_1$, $X_1\ dv'\ X_1$, $X_2\ dv\ X_2$ and $X_2\ dv'\ X_2$) and no other relations hold between X_1 and X_2. This gives us $X_2 = g(S', dv)$ and $X_1 = g(S', dv')$. So, our assumption of (i) is not invalidated, but, $dv \cap dv' = \{(X_1, X_1), (X_2, X_2)\}$, so both X_1 and X_2 are maximal elements by $dv \cap dv'$ and neither are greatest elements. Thus, $g(S', dv \cap dv') =\perp$, and S' is the required counterexample. □

So, in the same way as we struggled to push well foundedness solely into development we are struggling to push the existence of greatest elements solely into the constituent development relations. The following shows that the strong condition that we finally used to do this in the previous section does not work here.

Proposition 23
S is finite and non-empty $\not\Longrightarrow g(S, dv) \neq\perp$.

Proof
Consider the set S' used as the counterexample in the last propositon, 22; S' is finite but has no greatest element.
□

So, enforcing finiteness of development sets cannot guarantee the existence of greatest elements in unification sets. We are left then with a smaller set of well behavedness properties for this section.