Adequacy-preserving transformations of COSY path programs

M. Koutny

Abstract

We here investigate a transformation of COSY path programs which is a replacement of a single event in one path program $Q$ by another path program $P$. Such a transformation, called insertion, closely corresponds to refining an atomic action into several more elementary actions. In general, the path program resulting from the insertion of $P$ into $Q$ has quite different behavioural properties than do $P$ and $Q$. Therefore, in order to be able to use insertions as a verification tool it is necessary to develop a set of rules which would guarantee the preservation of various important properties through the insertion. One of such properties is adequacy which is a property capturing the absence of partial deadlocks in a system. In this paper we present some sufficient conditions for adequacy-preserving insertion. Also, we fully characterise the set of all those programs $P$ for which insertion is always adequacy-preserving.

Series Editor: M.J. Elphick

© 1988 University of Newcastle upon Tyne,
Printed and published by the University of Newcastle upon Tyne,
Computing Laboratory, Claremont Tower, Claremont Road,
Newcastle upon Tyne, NE1 7RU, England.
Bibliographical details

KOUTNY, Maciej

Adequacy - preserving transformations of COSY path programs. [By] M. Koutny

Newcastle upon Tyne: University of Newcastle upon Tyne: Computing Laboratory, 1988.

(University of Newcastle upon Tyne, Computing Laboratory, Technical Report Series, no. 270)

Added entries

UNIVERSITY OF NEWCASTLE UPON TYNE.
Computing Laboratory. Technical Report Series. 270

Abstract

We here investigate a transformation of COSY path programs which is a replacement of a single event in one path program Q by another path program P. Such a transformation, called insertion, closely corresponds to refining an atomic action into several more elementary actions. In general, the path program resulting from the insertion of P into Q has quite different behavioural properties than do P and Q. Therefore, in order to be able to use insertions as a verification tool it is necessary to develop a set of rules which would guarantee the preservation of various important properties through the insertion. One of such properties is adequacy which is a property capturing the absence of partial deadlocks in a system. In this paper we present some sufficient conditions for adequacy-preserving insertion. Also, we fully characterise the set of all those programs P for which insertion is always adequacy-preserving.

About the author

Dr. M. Koutny has been at the Computing Laboratory as a Research Associate from March, 1985 until Nov. 1986 and as a Lecturer from December, 1986. Prior to this he was at Warsaw Technical University in Poland from 1982.

Suggested keywords

ADEQUACY
ATOMIC ACTIONS
CONCURRENT PROGRAM VERIFICATION
CONCURRENT SYSTEMS
COSY MODEL
DISTRIBUTED SYSTEMS
PATH EXPRESSIONS

Suggested classmarks (primary classmark underlined)

Dewey (18th): 001.64404 001.6425
U.D.C. 681.322.06 519.687
Adequacy-Preserving Transformations of COSY Path Programs

Maciej Koutny
Computing Laboratory
The University
Newcastle upon Tyne NE1 7RU, U.K.

ABSTRACT

We here investigate a transformation of COSY path programs which is a replacement of a single event in one path program $Q$ by another path program $P$. Such a transformation, called insertion, closely corresponds to refining an atomic action into several more elementary actions. In general, the path program resulting from the insertion of $P$ into $Q$ has quite different behavioural properties than do $P$ and $Q$. Therefore, in order to be able to use insertions as a verification tool it is necessary to develop a set of rules which would guarantee the preservation of various important properties through the insertion. One of such properties is adequacy which is a property capturing the absence of partial deadlocks in a system. In this paper we present some sufficient conditions for adequacy-preserving insertion. Also, we fully characterise the set of all those programs $P$ for which insertion is always adequacy-preserving.

1 INTRODUCTION

The verification of the behavioural properties of concurrent systems is usually a difficult and complex task, and a number of techniques has been developed to make this task easier (see, for instance, [Cla86], [Hen85], [Mur81] and [Old86]). In this paper we investigate a technique which has been proposed as a means for verification of COSY path programs [Lau84]. In [Shi79] Shields introduced a transformation (called insertion) of path programs which consists in a replacement of a single event $\omega$ in one path program $Q$ by another path program $P$, to give a new path program, denoted $P \rightarrow_\omega Q$. Such a transformation closely corresponds to refining an atomic action into several more elementary actions which is a common technique in the development of concurrent programs [Dij78 and Lam77].

In this paper we discuss conditions under which insertion is an adequacy-preserving operation (adequacy is a property which captures the absence of partial deadlocks in a system). More precisely, we ask what conditions must be satisfied by $P$ and $Q$ in order that ($Q$ is adequate) $\iff (P \rightarrow_\omega Q$ is adequate).

The study of adequacy-preserving insertions was initiated by Shields [Shi79] and was then continued in [Hil83] and [Kou87]. In this paper we first recall the basic result of [Shi79] and present its possible generalization. This gives us sufficient conditions for adequacy-preserving insertion which we then use to obtain a characterisation of universally smooth path programs, i.e. those programs $P$ for which the insertion is always adequacy-preserving.

† A short version of this report will appear in the Lecture Notes in Computer Science, 1988.
The paper is organised as follows. In the next section we present a brief introduction to the basic COSY notation. In the third section we introduce sufficient conditions for adequacy-preserving insertion - a fundamental result from the point of view of the subsequent development. In the fourth section we provide a full characterisation of universally smooth path programs.

2 A BRIEF INTRODUCTION TO BASIC COSY NOTATION

COSY (CONcurrent SYstems) is a formalism intended to simplify the study of synchronous aspects of concurrent systems by abstracting from all aspects of systems except those which have to do with synchronization. A basic COSY path program \( P = \phi_1 \ldots \phi_n \) is a finite sequence of paths \( \phi_i \), each path being a regular expression enclosed between path and end parentheses, e.g.

\[
P = \text{path } a; b, c \text{ end}
\]

\[
\text{path } (d, e)^*; b \text{ end.}
\]

In such path expressions, the semicolon denotes sequence (concatenation), and the comma denotes mutually exclusive choice. The comma binds more tightly than semicolon and thus the expression \( a; b, c \) means "first \( a \) then \( b \) or \( c \)". An expression may be enclosed in conventional parentheses, and if the Kleene star is appended this means that the enclosed expression may be repeated zero or more times. The entire expression enclosed between path and end is implicitly "starred". A formal description of the COSY syntax may be found in [Lau84], [Lau81] and [Shi79].

The semantics of path programs can be defined in terms of vectors of strings which is an approach introduced in [Shi79].

With each path \( \phi \) is associated its set of events, denoted \( Ev(\phi) \), being the set of all symbols constituting the regular expression enclosed between path and end; and its set of cycles, denoted \( Cyc(\phi) \), being the regular language generated by this expression.

E.g., if \( \phi = \text{path } a; b, c \text{ end} \) then \( Ev(\phi) = \{a, b, c\} \) and \( Cyc(\phi) = \{ab, ac\} \).

The set of firing sequences of \( \phi \), denoted \( FS(\phi) \), is the set of all prefixes of the strings of \( Cyc(\phi)^* \).

The set of firing sequences \( FS(\phi) \) models the sequential behavior of a subsystem represented by path \( \phi \). To model non-sequential behaviour of program \( P = \phi_1 \ldots \phi_n \) we will employ partial orders of event occurrences represented by vectors of strings. A vector of strings \([v_1, \ldots, v_n]\) will be a possible history of \( P \) if each \( v_i \) is a possible history (firing sequence) of path \( \phi_i \), and if the \( v_i \)'s agree about the number and order of occurrences of events shared by the paths.

Let \( Evp = \bigcup_{\phi \in \mathcal{F} \mathcal{P}} Ev(\phi) \). For every \( v \in Evp^* \), \( v \) is the vector of strings \([v_1, \ldots, v_n] \), where each \( v_i \) is obtained from \( v \) by deleting events not belonging to \( Ev(\phi_i) \). The concatenation of two vectors of strings \( w = [w_1, \ldots, w_n] \) and \( u = [u_1, \ldots, u_n] \), denoted \( uw \), is the vector \([w_1u_1, \ldots, w_nu_n] \).

Let \( Vep = \{w : a \in Evp\} \). The set of all possible histories (or vector firing sequences) of \( P \), denoted \( VFS_P \), is defined as \((FS(\phi_1) \times \cdots \times FS(\phi_n)) \cap Vep^* \).

A number of results has been obtained concerning both the theory and application of the vector firing sequence semantics. A review of most of these results can be found in [Lau84]. A general theory of vector firing sequences was developed in [Shi85].

**NOTATION 2.1**

Let \( P \) be a path program, \( v \in Evp^* \) and \( V \subseteq Vep^* \).

1. \( |P| \) is the number of paths constituting \( P \).
2. $P(i)$ is the $i$-th path constituting $P$.
3. $VCycP = (Cyc(P(1)) \times \ldots \times Cyc(P(n))) \cap Veuv^*$ (note: $VCycP \subseteq VFS_P$).
4. For every event $a$, $\#_a(\mu) = \#_a(\nu)$ is the number of occurrences of $a$ within $\nu$.
5. $Ev(\nu) = \{a: \#_a(\mu) \geq 1\}$.
6. $Ev(V) = \bigcup_{\nu \in V} Ev(\nu)$ and Pref($V$) = $\{w \in Veuv^*: w t \in V$ for some $t \in Veuv^*\}.
7. $\gamma$ is a prefix of a vector $\nu$, denoted $\nu \preceq \gamma$, if $\gamma \in Pref(\nu).
8. $V$ is a prefix of $W \subseteq Veuv^*$, denoted $V \preceq W$, if $V \subseteq Pref(W).
9. A sequence of strings $\mu = (v_1, \ldots, v_k), k \geq 0$, is a $P(i)$-decomposition of $\nu$ if
   a. $\nu = v_1 \ldots v_k$ (if $k = 0$ then $v_1 \ldots v_k = \varepsilon$, where $\varepsilon$ denotes the empty string).
   b. $v_1, \ldots, v_k \in Cyc(P(i))$.
   c. $v_k \in Cyc(P(i))$ or $v_k$ is a non-empty prefix of a string of $Cyc(P(i))$.
   Furthermore, if $k = 0$ or $v_k \in Cyc(P(i))$ then $\mu$ is proper.
10. A sequence of vectors $\nu = (\gamma_1, \ldots, \gamma_k), k \geq 0$, is a $P$-decomposition of $\gamma$ if
    a. $\nu = \gamma_1 \ldots \gamma_k$ (if $k = 0$ then $\gamma_1 \ldots \gamma_k = \varepsilon$).
    b. $\gamma_1, \ldots, \gamma_k \in VCycP$.
    c. $\gamma_k \in VCycP$ or $\gamma_k \in Pref(VCycP)$.
    Furthermore, if $k = 0$ or $\gamma_k \in VCycP$ then $\nu$ is proper.

The property of path programs we will refer to throughout this paper is adequacy which captures the absence of partial deadlocks in a system. We say that a path program $P$ is **adequate** if for every $\nu \in VFS_P$ there is a $\gamma \in Veuv^*$ such that $\nu \gamma \in VFS_P$ and $Ev(\gamma) = EvP$.

Finally, for every vector $\rho = \{\rho_1, \ldots, \rho_n\}$, we will denote by $\rho_i$ its $i$-th co-ordinate, $\rho_i$.

The $\underline{\gamma}$ (underline) notation may be ambiguous if, for instance, $\nu \in (EvP \cap EvQ)^*$.

All such ambiguities will be resolved by the context.

3 Insersion of Path Programs

Insertion of path programs [Shi79, Hil83 and Kou87] is a replacement of a single event in one path program by another path program. In order to carry out an insertion we need two programs $P$ and $Q$ such that the number of paths constituting program $P$ is the same as the number of paths in program $Q$ which employ a distinguished event $\omega$. We then substitute the regular expression of the $i$-th path of program $P$ for every occurrence of event $\omega$ in the $i$-th path of $Q$ employing this event. The _resulting_ program will be called the **insertion** of $P$ into $Q$ and denoted by $P \rightarrow_i Q$ or simply $P \rightarrow Q$.

In this section we will present conditions under which insertion is an adequacy-preserving transformation, i.e. the adequacy of $Q$ implies adequacy of $P \rightarrow Q$, and vice-versa. We start by introducing some basic definitions, and then derive a general result providing sufficient conditions for adequacy-preserving insertions.

Let $\Pi$ be the set of all programs $P$, and let $\Delta$ be the set of all programs $Q$ such that
a. $EvP \subseteq \Sigma$ and $EvQ \subseteq \Omega$.

b. $n = |P| \leq |Q|$.

c. $\omega \in \bigcap_{i \leq n} Ev(Q(\mu))$ and $\omega \in \bigcup_{i \geq n} Ev(Q(\mu))$.

where $\Sigma$ and $\Omega$ are two fixed disjoint sets of events and $n \geq 2$ is a fixed integer. We will use $A, B, C, \ldots$ to denote the events of $\Sigma$, and $a, b, c, \ldots$ to denote the events of $\Omega$ other than $\omega$. 
DEFINITION 3.1

For \( P \in \Pi \) and \( Q \in \Delta \), let \( P \rightarrow Q \) denote the program \( P \rightarrow Q = R_1 \ldots R_n \) such that each path \( R_i \) is derived from \( Q(i) \) by substituting the regular expression of \( P(i) \) enclosed in parentheses for every occurrence of \( \omega \).

Throughout this paper we will denote \( E_{uv \rightarrow q} \) by \( Ev \); \( V_{uv \rightarrow q} \) by \( Vev \); and \( VFS_{P \rightarrow Q} \) by \( VFS \).

Consider the following two path programs.

\[
P_1 = \begin{array}{l}
\text{path } A;B \\
\text{path } A;C
\end{array}
\]

\[
Q_1 = \begin{array}{l}
\text{path } \omega;A;\omega;B;\omega,A \\
\text{path } a;b;\omega;A
\end{array}
\]

After inserting \( P_1 \) into \( Q_1 \) we obtain

\[
P_1 \rightarrow Q_1 = \begin{array}{l}
\text{path } ((A;B);a);b;((A;B);a) \\
\text{path } a;b;((A;C);a)
\end{array}
\]

The above is an example of adequacy-preserving insertion (both \( Q_1 \) and \( P_1 \rightarrow Q_1 \) are adequate programs). The situation changes if instead of \( P_1 \) we take

\[
P_2 = \begin{array}{l}
\text{path } B;A \\
\text{path } C;A
\end{array}
\]

since \( P_2 \rightarrow Q_1 \) is not even deadlock-free (e.g. \( B \) leads to a deadlock).

DEFINITION 3.2

1. For every \( w \in Ev^* \), \( w|_P \) is the string obtained from \( w \) by deleting events not belonging to \( \Sigma \).
2. For every \( u = [u_1, \ldots, u_n] \in Ev^* \), \( u|_P \) is the vector \( [u_1|_P, \ldots, u_n|_P] \).

We remark that \( u|_P \) can be thought of as being a projection the vector \( u \) onto \( P \).

PROPOSITION 3.3

\( Vev^*|_P \subseteq V_{uv} \subseteq VFS |_P \subseteq VFS_P \).

PROOF

Elementary, e.g. by induction on the length of a vector of strings \( l(w) \), defined in a standard way by \( l(\omega) = 1 + l(w) \).

The next result is us a simple characteristic of firing sequences of \( P \rightarrow Q \) in terms of firing sequences of \( P \) and \( Q \).

PROPOSITION 3.4

A string \( w \in Ev^* \) is a firing sequence of \( (P \rightarrow Q)_i \), \( i \leq n \), if and only if there is a sequence of strings \( (t_1, \ldots, t_k, v_1, \ldots, v_{k+1}) \), \( k \geq 0 \), such that

a. \( v_1\omega \ldots v_{k+1} \) is a firing sequence of \( Q(i) \) such that \( \#_\omega(v_1 \ldots v_{k+1}) = 0 \).

b. \((t_1, \ldots, t_k)\) is a \( P(i) \)-decomposition of \( w|_P \), and if \( v_{k+1} \neq \omega \) then \((t_1, \ldots, t_k)\) is proper.

c. \( w = v_1 t_1 \ldots v_k t_k v_{k+1} \).

PROOF

Follows directly from the definition of \( P \rightarrow Q \).

To formulate the main result of [Shi79] and subsequently its generalisation, we need a definition of the set, denoted \( CVFS \), of those vector firing sequences of program \( P \rightarrow Q \) which can be derived from vector firing sequences of program \( Q \) by replacing every occurrence of the vector \( \omega \) by a "cyclic" vector firing sequence of program \( P \), i.e. one belonging to \( VCyc_P \).
**Definition 3.5**

A *chopping* of \( w \in Vev^* \) is a sequence of vectors \((l_1, \ldots, l_k, u_1, \ldots, u_k + 1)\), \( k \geq 0 \), such that

a. \( u_1 \omega_1 \ldots u_{k+1} \omega_{k+1} \) is a history of \( Q \).

b. \((l_1, \ldots, l_k)\) is a \( P \)-decomposition of \( w/\ell_p \).

c. \( w = u_1 \ell_1 \ldots u_k \ell_k + 1 \).

**Definition 3.6**

For every vector \( w \in Vev^* \), \( w/Q \) is the set of all histories \( u \in VFS_Q \) for which there is at least one chopping \((l_1, \ldots, l_k, u_1, \ldots, u_k + 1)\) of \( w \) such that \( u = u_1 \omega_1 \ldots u_{k+1} \omega_{k+1} \) and \((l_1, \ldots, l_k)\) is a proper \( P \)-decomposition of \( w/P \).

Let \( CVFS \) denote the set of all vectors \( w \in Vev^* \) such that \( w/Q \neq \emptyset \).

It turns out that \( CVFS \) is a subset of \( VFS \). We remark that \( w/Q \) may contain infinitely many elements, e.g. if \( g \in VCycp \) and \((\omega)^* \subseteq VFS_Q \) then \( card(g/Q) = \infty \).

**Proposition 3.7**[[Shi79]]

\( CVFS \subseteq VFS \).

**Proof**

Follows directly from Proposition 3.4.

**Proposition 3.8** (Substitution Lemma of [[Shi79]])

Let \( P \in II \) and \( Q \in \Delta \).

If \( P \) is adequate, \( VFS = Pref(CVFS) \) and \( VFS_Q = Pref(VCycp^*) \) then

\( (Q \) is adequate) \( \Rightarrow \) \( (P \rightarrow Q \) is adequate).

The restriction imposed on \( P \) in the Substitution Lemma can be slightly relaxed.

**Lemma 3.9**

Let \( P \in II \) and \( Q \in \Delta \).

If \( VFS = Pref(CVFS) \) and \( Ev(VCycp) = EvP \) then

\( (Q \) is adequate) \( \Rightarrow \) \( (P \rightarrow Q \) is adequate).

The first condition essentially requires that the histories of \( P \rightarrow Q \) be histories of \( Q \) with all the \( \omega \)'s being replaced by some cyclic histories of \( P \). This results in the adequacy of \( P \rightarrow Q \) w.r.t. the events originating from \( Q \), and also guarantees that after any history of \( P \rightarrow Q \) we can eventually execute an arbitrary cyclic history of \( P \). The latter property together with the second condition ensures the adequacy of \( P \rightarrow Q \) w.r.t. events originating from \( P \).

**Proof of Lemma 3.9**

By \( VFS = Pref(CVFS) \), it suffices to show that if \( w \in CVFS \) then there is \( wz \in CVFS \) with \( Ev(z) = Ev \).

Let \( u \) and \((l_1, \ldots, l_k, u_1, \ldots, u_k + 1)\) be as in Definition 3.6. Furthermore, let \( z_1, \ldots, z_k \) be any vectors of \( VCycp \) such that \( EvP = Ev(z_1 \ldots z_i) \). (Such vectors can be found since \( Ev(VCycp) = EvP \) and \( EvP \) is finite.) By the adequacy of \( Q \), there is \( u_1 \omega_1 \ldots u_{k+1} \omega_{k+1} + 1 \ell_k \omega_k + 1 \ell_m \omega_m + 1 \in VFS_Q \) such that

\( Ev(u_1 u_2 \ldots u_{k+1}) = EvQ\{\omega\} \) and \( m \geq l \).

Let \( u = u_1 \ell_1 \ldots u_k \ell_k + 1 u_1 \ell_1 \ldots u_m \ell_m \ell_{m+1} \), where \( z_1 = z_1 + 1 = \ldots = z_m \).

We have \( u = wz \in CVFS \) and \( Ev(z) = Ev \), which completes the proof.
The implications in Proposition 3.8 and Lemma 3.9 cannot be reversed. To show this we consider the following path programs.

\[ P_3 = \text{path } A^* \text{ end} \]
\[ Q_3 = \text{path } (\omega; \omega; \alpha), (\omega; c; \alpha; b) \text{ end} \]
\[ P_3 \rightarrow Q_3 = \text{path } ((A^*); (A^*); a); ((A^*); c; \alpha; b) \text{ end} \]
\[ \text{path } ((A^*); (A^*); b); ((A^*); c; \alpha; b) \text{ end} \]

Although \( P_3 \) and \( Q_3 \) satisfy all the conditions in Proposition 3.8 and Lemma 3.9, \( P_3 \rightarrow Q_3 \) is not an adequacy-preserving insertion. For \( P_3 \rightarrow Q_3 \) is an adequate program whereas \( Q_3 \) is not (e.g. \( \omega \omega \) leads to a deadlock). To be able to reverse the implication in Lemma 3.9 we have to require a stronger relationship between the histories of programs \( P \rightarrow Q \) and \( Q \).

**Lemma 3.10**

Let \( P \in \Pi \) and \( Q \in \Delta \) be such that

1. \( VFS = \text{Pref}(CVFS) \)
2. If \( w \preceq u \) then \( u|Q \preceq u|Q \), for all \( w, u \in CVFS \).

Then: (\( P \rightarrow Q \) is adequate) \( \Rightarrow \) (\( Q \) is adequate). \( \square \)

We first note that \( P_3 \) and \( Q_3 \) do not satisfy \((*)\). Indeed, if we take \( u = AAa \) and \( u = AACc \) then \( u|Q_3 = (\omega; \omega; \omega) \) and \( u|Q_3 = (\omega; \omega; c) \). The role of \((*)\) might be explained in the following way. Suppose that we are about to demonstrate that it is possible to execute an event \( c \) after a history \( t \) of program \( Q \). Knowing that \( P \rightarrow Q \) is an adequate program, perhaps the most natural way of demonstrating this would be to show that the following three-step procedure can be successfully carried out.

1. Expand \( t \) to any history \( u \in CVFS \) such that \( t \in u|Q \).
2. Find any \( u \in CVFS \) such that \( c \) is an event in \( z \).
3. Contract \( u \) to any \( z \in u|Q \) such that \( t \) is a prefix of \( z \).

Clearly, (1) and (2) can always be carried out (step (2) succeeds since \( P \rightarrow Q \) is adequate and \( VFS = \text{Pref}(CVFS) \)). The same may be not true of step (3) and we have just seen an example of such a situation. If, however, \((*)\) holds then (3) always succeeds.

**Proof of Lemma 3.10**

We observe that if \( \bigcup_{1 \leq n} Ev(Q_{11}) = \{\omega\} \) then the lemma trivially holds. Hence, without loss of generality, we may assume that there is \( r \in Ev(Q_{11}) \) such that \( r \neq \omega \). Also, by the adequacy of \( P \rightarrow Q \) and \( VFS = \text{Pref}(VCFS) \), \( VCycp \neq \emptyset \).

Let \( t \in VFS \) and \( \theta \) be any event of \( Ev(P_{11}) \). By \( VCycp \neq \emptyset \), there is \( u \in CVFS \) such that \( t \in u|Q \).

Consequently, by the adequacy of \( P \rightarrow Q \) and \( VFS = \text{Pref}(CVFS) \), there is \( u = uw \in CVFS \) such that \( Ev_Q - (\omega) \subseteq Ev(z) \) and \( z \) can be represented as

\[ z = \Upsilon \tau \Upsilon \tau \cdots \Upsilon \tau \Upsilon \tau_{m+1} \text{ where } m > \#_u(t) \).

Hence, by \( \tau \in Ev(Q_{11}) \) and \( \theta \in Ev(P_{11}) \), we have \( \#_u(z) \geq m > \#_u(t) \), for all \( z \in u|Q \).

By \( u|Q \preceq u|Q \), there is \( z \in w|Q \) such that \( t \ll u \). By \( Ev_Q -(\omega) \subseteq Ev(z) \) and \( \#_u(z) > \#_u(t) \), we have \( \#_t(t) < \#_t(u) \), for all \( t \in Ev_Q \). This completes the proof. \( \square \)

By joining Lemma 3.9 and 3.10 we obtain the main result of this section.
Theorem 3.11

Let \( P \in \Pi \) and \( Q \in \Delta \) satisfy the following.
1. \( \text{VFS} = \text{Pref}(\text{CVFS}) \).
2. \( \text{Ev}(\text{VCycp}) = \text{EvP} \).
3. If \( w \preceq y \) then \( \frac{w}{Q} \preceq \frac{y}{Q} \), for all \( w,y \in \text{CVFS} \).

Then: \( (P \rightarrow Q \text{ is adequate}) \Leftrightarrow (Q \text{ is adequate}) \). □

The reciprocal theorem does not hold. To show this we consider the following path programs.

\[ P_4 = \begin{array}{l}
\text{path } A^*, (B;B) \end{array} \quad Q_4 = \begin{array}{l}
\text{path } (\omega; \omega; \omega; \omega; \omega; \omega) \end{array} \]

We observe that neither \( Q_4 \) nor \( P_4 \rightarrow Q_4 \) is an adequate program, hence the insertion is an adequacy-preserving one. On the other hand, as one may easily check, none of the conditions in Theorem 3.11 is satisfied.

4 Universally Smooth Path Programs

The last result of the previous section provides us with a means of proving the adequacy-preservation for a number of interesting insertions. In this section we will use Theorem 3.11 to characterise the universally smooth path programs, i.e. those programs \( P \) for which insertion is always adequacy-preserving.

We will say that a path program \( P \) is smooth w.r.t. a non-empty set \( \Gamma \) of path programs, \( P \in \text{Smooth}(\Gamma) \), if \( (Q \text{ is adequate}) \Leftrightarrow (P \rightarrow Q \text{ is adequate}) \) for all \( Q \in \Gamma \). The universally smooth programs are programs in \( \text{Smooth}(\Delta) \).

To justify the need for the conditions to be satisfied by universally smooth programs we will analyse in more detail two examples of insertions which are not adequacy-preserving.

\[ P_5 = \phi_1 \phi_2 = \begin{array}{l}
\text{path } A; (A;B) \end{array} \quad Q_5 = \begin{array}{l}
\text{path } \omega; \omega; b \end{array} \]

\[ P_5 \rightarrow Q_5 = \begin{array}{l}
\text{path } (A;A); a; b \end{array} \quad \text{end} \]

Clearly, \( Q_5 \) is adequate whereas \( P_5 \rightarrow Q_5 \) is not, e.g. \( \bar{x} = ABa \) leads to a deadlock. We observe that the "projection" of such an \( \bar{x} \) onto \( P_5 \) results in the vector firing sequence \( \bar{u} = AB \) such that: (i) \( \phi_1 \) completed one full cycle, \( AB \in \text{Cyc}(\phi_1) \); (ii) \( \phi_2 \) did not complete the cycle it began, \( A \in \text{Cyc}(\phi_2) \); and (iii) to complete the cycle in \( \phi_2 \) it is necessary to execute event \( A \) requiring a participation of \( \phi_1 \). In the case of our example (iii) cannot be accomplished and thus the cycle began by \( \phi_2 \) will never be completed.

A universally smooth path program has therefore to satisfy the following: For every history \( \bar{u} \) it must be possible to complete cycles in all the paths without executing any event \( \omega \) in those paths which have already completed their cycles.

\[ P_6 = \psi_1 \psi_2 = \begin{array}{l}
\text{path } A; (A;A) \end{array} \quad Q_6 = \eta_1 \eta_2 = \begin{array}{l}
\text{path } (\omega; \omega; \omega; \omega; \omega; \omega) \end{array} \]

\[ P_6 \rightarrow Q_6 = \begin{array}{l}
\text{path } (A;A); a; b; c; a \end{array} \quad \text{end} \]

Although \( P_6 \) satisfies the above requirement, the insertion \( P_6 \rightarrow Q_6 \) is not adequacy-preserving.

For \( Q_6 \) is adequate whereas \( P_6 \rightarrow Q_6 \) is not, e.g. \( AAa \) leads to a deadlock. The reasons why we are getting into troubles this time might be explained in the following way.
Having executed $A\mathbf{Aq}$ one might say that $q_1$ completed one full cycle, $A\in C_{\text{yc}}(q_1)$, which intuitively corresponds to the execution of $\omega$ in $q_1$, whereas $q_2$ completed two full cycles, $A\in C_{\text{yc}}(q_2)C_{\text{yc}}(q_2)$, which corresponds to the execution of $\omega\omega$ in $q_2$. Therefore $A\mathbf{Aq}$ corresponds to the "execution" of vector $[\omega, \omega\omega]$ in $Q_d$ leading to a deadlock. Consequently, it must be the case that for every history of a universally smooth program the number of cycles performed by the paths constituting the program is the same for all the paths.

**Definition 4.1**

Let $\Pi_o \subseteq \Pi$ denote the set of all programs $P$ such that

1. $E_{P}(V_{CycP}) = E_{P}p$.
2. For every $u \in P_{\text{rc}}(V_{CycP})$, there is $u\in V_{CycP}$ such that for all $i \leq n$,
   
   $u|_i \in C_{\text{yc}}(P_{(i)}) \Rightarrow u|_i = e$.
3. For every $u \in V_{FS}P$ and all $i, j \leq n$, if $\mu$ is a $P_{(i)}$-decomposition of $u|_i$ and $\gamma$ is a $P_{(j)}$-decomposition of $u|_j$ then $\mu$ and $\gamma$ have the same length. \qed

We will now prove some of the properties of the programs of $\Pi_o$.

**Proposition 4.2**

If $P \in \Pi_o$ then $e \in C_{\text{yc}}(P_{(i)})$, for all $i \leq n$.

**Proof**

Follows immediately from Definition 4.1(3). \qed

**Proposition 4.3**

Let $P \in \Pi_o$, $u \in V_{FS}P$, and $k \geq 1$ be the common length of the decompositions of $u|_i$'s. Also, let $d_j = (d_{i_1}, ..., d_{i_k})$ be any $P_{(i)}$-decomposition of $u|_i$, for all $i \leq n$; and $h_j = (d_{i_1}, ..., d_{i_k})$, for $j = 1, ..., k$.

1. $h = (h_1, ..., h_k)$ is a $P$-decomposition of $u$.
2. $d_j = e$, for all $i$ and $j$.
3. $\text{first}(d_{i_1}) = \ldots = \text{first}(d_{i_k})$, for all $j \leq k$. (first(s) is the first symbol of a non-empty string s)
4. $d_1$ is the unique $P_{(i)}$-decomposition of $u|_1$, for all $i$.
5. $h$ is the unique $P$-decomposition of $u$.
6. There is $d \in V_{CycP}$ such that $d \in d$ and for all $i \leq n$,
   
   $d_{i_1} \in C_{\text{yc}}(P_{(i)}) \Rightarrow d_{i_1} = d_{i_k}$.

**Proof**

(1) We observe that it suffices to show that $h_j \in V_{\text{Ep}}$, for all $j \leq k$.

Suppose that $f = h_1 \ldots h_j \in V_{\text{Ep}}$, for some $j \leq k$. Let $s$ be a minimal prefix of $w$ such that $h_i$ is a prefix of $v|_j$, for all $i \leq n$. By the minimality of $u$, $v|_l \neq f|_l$ for some $l$. On the other hand, by $f \in V_{\text{Ep}} \exists w$, there is $m$ such that $v|_m = f|_m$, where $u \neq e$ is a prefix of $d_{m(i+1)} \ldots d_{mk}$. Hence $v|_l$ has a $P_{(j)}$-decomposition of length $j$, whereas $v|_m$ has a $P_{(m)}$-decomposition of a length greater than $j$, contradicting Definition 4.1(3).

Thus $h_1 \ldots h_j \in V_{\text{Ep}}$, for all $j \leq k$, which yields $h_j \in V_{\text{Ep}}$, for all $j \leq k$.

(2) and (3) From Proposition 4.2 it follows that $\lambda = (\ldots, \lambda)$, for every $\lambda \in E_{P}$ such that $\lambda \in V_{FS}$. This, (1) and Proposition 4.2 gives (2) and (3).

(4) Suppose that (4) is not satisfied for $i = 1$.

Without loss of generality, we assume that $(f_1, ..., f_k)$ is a $P_{(i)}$-decomposition of $u|_1$ such that $f_1 = d_{i_1}, ..., f_{i-1} = d_{i(i-1)}$ and $f_i \neq d_{i_1}$, for some $1 \leq i \leq k - 1$. 

...
From (1) it follows that both \( u = d_1 \) and \( t = [f_1, d_2, d_3, \ldots, d_m] \) belong to \( V_{\text{ep}}^* \) and from (2,3) it follows that first\((d_{i+1}) = \text{first}(d_{2i+1}) = \text{first}(d_{4i+1}) = \lambda \) (the \( d_i \)'s were arbitrary \( P_{(i)} \)-decompositions of the \( u_i \)'s!). Hence we derived a contradiction since:

\[
\#_1(u_i) = \#_2(u_i) = \#_3(u_i) = \#_4(u_i).
\]

(5) Follows from (4).
(6) Follows from \( h_k \in \text{Pref}(V_{\text{cyc}}) \) and Definition 4.1(2).

**COROLLARY 4.4**

Let \( P \in \Pi_0 \), \( u \in V_{\text{FS}} \) and \( e \preceq u \).

1. \( u \) has exactly one \( P \)-decomposition.
2. Each \( u_i \) has exactly one \( P_{(i)} \)-decomposition.
3. If \( (d_1, \ldots, d_k) \) is the \( P \)-decomposition of \( u \) then \( (d_1, \ldots, d_l, d) \) is the decomposition of \( u \), for some \( l \leq k \) and \( e \preceq d \preceq \).

The next result is proved in the Appendix.

**LEMMA 4.5**

Let \( P \in \Pi_0 \) and \( Q \in \Delta \).

1. \( V_{\text{FS}} \subseteq \text{Pref}(CV_{\text{FS}}) \).
2. If \( u \preceq u \) then \( u \preceq u \), for all \( u, v \in CV_{\text{FS}} \).

As one might have expected, the universally smooth programs is rather a small subset of \( \Pi \), e.g. if \( u \) is a non-empty history of \( P \) then all the \( u_i \)'s begin with the same event. Nevertheless, it is rather surprising that \( \text{Smooth}(\Delta) \) coincides with the sets \( \text{Smooth}(\Delta_0) \) and \( \text{Smooth}(\Delta_1) \) for two relatively small sets of programs, \( \Delta_0 \) and \( \Delta_1 \).

**DEFINITION 4.6**

1. Let \( \Delta_0 \) be the set of all programs \( Q \) such that

\[
Q = \begin{array}{ll}
\text{path} & \omega; a; a; \text{end} \\
\vdots \\
\text{path} & \omega; a; a; \text{end}
\end{array}
\]

or \( Q \) is composed of \( \psi_k \), \( \psi_k \) and \( n \cdot 2 \) copies of \( \xi \), where

\[
\xi = \begin{array}{ll}
\text{path} & \omega; a; \text{end}
\end{array}
\]

\[
\psi_l = \begin{array}{ll}
\text{path} & b; ((\omega; b); c; (\omega; a)) \text{ end}
\end{array}
\]

\[
\psi_k = \begin{array}{ll}
\text{path} & (\omega; a)^{k-1}; b; ((\omega; b); a); c \text{ end (for } k \geq 2) \end{array}
\]

\[
\phi_l = \begin{array}{ll}
\text{path} & a; c; (\omega; a) \text{ end}
\end{array}
\]

\[
\phi_k = \begin{array}{ll}
\text{path} & (\omega; a)^{k-1}; a; c \text{ end (for } k \geq 2) \end{array}
\]

2. Let \( \Delta_1 \) be defined in the same way as \( \Delta_0 \), with the following changes.

\[
\xi = \begin{array}{ll}
\text{path} & \omega \text{ end}
\end{array}
\]

\[
\psi_l = \begin{array}{ll}
\text{path} & b; ((\omega; b); a); c; \omega \text{ end}
\end{array}
\]

\[
\psi_k = \begin{array}{ll}
\text{path} & (\omega; a)^{k-1}; b; ((\omega; b); a); c \text{ end (for } k \geq 2) \end{array}
\]

\[
\phi_l = \begin{array}{ll}
\text{path} & a; c; \omega \text{ end}
\end{array}
\]

\[
\phi_k = \begin{array}{ll}
\text{path} & (\omega; a)^{k-1}; a; c \text{ end (for } k \geq 2) \end{array}
\]

**REMARKS:**

a. If \( p \) is a regular expression and \( m \geq 1 \) then \( p^m \) stands for the regular expression \( (p_1 ; \ldots ; p_m) \), where \( p_1 = \ldots = p_m = (p) \).
b. In the above definition, \( a, b, c, a_1, \ldots, a_n \) are all different events. \( \square \)

**Proposition 4.7**

All programs of \( \Delta_0 \) and \( \Delta_1 \) are adequate.

**Proof**

\[
\begin{align*}
VFS_Q &= Pref([a_1 \ldots a_n]^*) \\
VFS_{T_k} &= Pref([b_ac]^{k-1})^* \quad \text{and} \quad VFS_{T_k} = Pref(([b_ac]^{k-1})^*) \quad \text{for } k \geq 2 \\
VFS_{R_k} &= Pref([b_ac]^*) \quad \text{and} \quad VFS_{R_k} = Pref(([b_ac]^{k-1})^*) \quad \text{for } k \geq 2
\end{align*}
\]

where \( Q \) is the first program introduced in Definition 4.6(1), \( T_k = \psi_k \phi_k \xi \ldots \xi \in \Delta_0 \) and \( R_k = \psi_k \phi_k \xi \ldots \xi \in \Delta_1 \), for \( k \geq 1 \). \( \square \)

We will now formulate and prove the main result of this paper which gives a full characterisation of the universally smooth path programs.

**Theorem 4.8**

\( \Pi_0 = \text{Smooth}(\Delta) = \text{Smooth}(\Delta_0) = \text{Smooth}(\Delta_1) \).

**Proof**

Let \( P \in \text{Smooth}(\Delta_0) \). We will show that \( P \in \Pi_0 \).

Let \( Q \) be the first program introduced in Definition 4.6(1). By the adequacy of \( P \to Q \) (Proposition 4.7), there is \( w = uv \in VFS \) such that \( E_{uv} = E_{u,v} \). Clearly, \( w|p \in VFC_p^* \), which yields Definition 4.1(1).

Suppose \( w \in VFC_{D_p} \). Then \( \iota = w a_1 \ldots a_n \in VFS \), where \( \{a_1, \ldots, a_n\} = K = \{k: w|k \in \text{Cycle}(P(k)) \} \). By the adequacy of \( P \to Q \) (Proposition 4.7), there is \( \tau \in VFS \) such that \( \#_a(\tau) = 0 \). We observe that \( \tau|p \in VFC_p, w \equiv (\tau|p)|p \) and \( w|k = (\tau|p)|k \), for all \( k \in K \). Hence Definition 4.1(2) holds.

Suppose now that Definition 4.1(3) is not satisfied for some \( w \in VFS \). Without loss of generality, we may assume that \( w|1 \) has a \( P(1) \)-decomposition \( (d_1, \ldots, d_k) \) and \( w|2 \) has a \( P(2) \)-decomposition of a length \( m < k \).

Let \( Q = \psi_k \phi_k \xi \ldots \xi \in \Delta_0 \). We observe that \( x = [d, w|2, \ldots, w|a] \in VFS \), where \( d = d_1 \ldots d_k bb \) if \( d_k = c \), and \( d = d_1 \ldots d_k, bb \) if \( d_k = c \) (if \( k = 1 \) then \( d_1 = x \)). We have \( \#_c(x) = 0 = \#_d(y) \), for all \( x, y \in VFS((P \to Q), x) \) and \( x, y \in VFS((P \to Q), y) \). Hence it is not possible to execute the events \( a \) and \( c \) in the path \( x \), which contradicts the adequacy of \( P \to Q \). Thus Definition 4.1(3) is satisfied.

This completes the proof of \( \text{Smooth}(\Delta_0) \subseteq \Pi_0 \).

In a very similar way we may show that \( \text{Smooth}(\Delta_1) \subseteq \Pi_0 \). We then observe that \( \Pi_0 \subseteq \text{Smooth}(\Delta) \) follows from Theorem 3.11, Lemma 4.5 and Definition 4.1(1). Hence we obtain: \( \text{Smooth}(\Delta_0) \subseteq \Pi_0 \subseteq \text{Smooth}(\Delta) \) and \( \text{Smooth}(\Delta_1) \subseteq \Pi_0 \subseteq \text{Smooth}(\Delta) \).

This and \( \text{Smooth}(\Delta_0) \supseteq \text{Smooth}(\Delta) \subseteq \text{Smooth}(\Delta_1) \) yields the thesis. \( \square \)

The form of \( \omega \)-conflicts (i.e. conflicts involving \( \omega \)) being admitted by the programs of \( \Delta_0 \) and \( \Delta_1 \) is rather restrictive. Indeed, if \( Q \in \Delta_0 \) and \( x_\omega, x_\beta \in \text{FS}(Q(\omega)) \) then \( \beta = \omega \) or \( \beta = a \). If \( Q \in \Delta_1 \) then \( \omega \)-conflict is specified by exactly one path of \( Q \) and it is the unique conflict specified by \( Q \). Consequently, we may quite safely assert that for the majority of \( \Gamma \)'s containing programs which admit \( \omega \)-conflicts, \( \text{Smooth}(\Gamma) = \text{Smooth}(\Delta) \). A question should therefore be asked whether this will change if we assume that the programs of \( \Gamma \) exclude \( \omega \)-conflicts.

Let \( \Delta_{\text{occ}} \) denote the set of all \( Q \in \Delta \) which are \( \omega \)-conflict-free, i.e. if \( x_\omega \) and \( x_\beta \) belong to \( \text{FS}(Q(\omega)) \) then \( \omega = \beta \). It can be shown ([Kou87]) that \( \text{Smooth}(\Delta_{\text{occ}}) = \text{Smooth}(\Delta) \). Furthermore, we can try to
provide a characterisation of $\text{Smooth}(\Delta_{\omega^k})$ in a similar style as we did for universally smooth programs.

**Definition 4.9**

Let $\Pi_1 \subseteq \Pi$ denote the set of all programs $P$ such that
1. $E(v) = E(v_P)$.
2. For every $\mu \in \text{Pref}(VCycP_k)$, $k \geq 1$, there is $\mu \in VCycP_k$ such that for all $i \leq n$,
   \[ \mu_i \in Cyc(P_{(j)})^k \Rightarrow \nu_i = e. \]
3. For every $\nu \in VFS_P$ and all $i,j \leq n$, if $\mu$ is a proper $P_{(j)}$-decomposition of $\nu_i$ and $\gamma$ is a proper $P_{(j)}$-decomposition of $\nu_j$, then $\mu$ and $\gamma$ have the same length. \qed

**Definition 4.10**

Let $\Delta_2$ be the set of all programs $Q$ such that for some $k \geq 1$

\[ Q = \text{path } \omega^k; a_1; a \text{ end} \]

\[ \ldots \]

\[ \text{path } \omega^k; a_m; a \text{ end} \]

or there are $k > m \geq 0$ such that $Q$ is composed of $\psi_k$, $\phi_{km}$ and $n\cdot 2$ copies of $\xi$, where

\[ \xi = \text{path } \omega \text{ end} \]

\[ \psi_k = \text{path } \omega^k; (b; (c; c)); c; d \text{ end} \]

\[ \phi_{k0} = \text{path } (a; (c; d)); \omega^k; c; d \text{ end} \]

\[ \phi_{km} = \text{path } \omega^m; (a; (c; d)); \omega^{k-m}; c; d \text{ end} \] (for $m > 0$) \qed

**Theorem 4.11**

$\text{Smooth}(\Delta_{\omega^k}) \subseteq \text{Smooth}(\Delta_2) \subseteq \Pi_1$.

**Proof**

Similar to the corresponding part in the proof of Theorem 4.8. \qed

We conjecture that both inclusions in Theorem 4.7 can be reversed. In any case, the programs in $\text{Smooth}(\Delta_{\omega^k})$ satisfy quite strong conditions, not very different from those characterising universally smooth path programs.

5 **Concluding Remarks**

In [Shi79] and [Hil83] the discussion focused on investigating various notions of connectness defined for path programs. As it was demonstrated in [Kou87], the connectness criteria are often too weak to force the different paths of a program to perform the same number of cycles (a property captured by Definition 4.1(3) and 4.9(3)) which seems to be a crucial property for the whole problem of adequacy-preserving insertions. We believe that the future research in this area should concentrate on investigation of programs which can be smoothly inserted into syntactically defineable classes of programs, and on generalisation of the results presented in the Section 3.
ACKNOWLEDGEMENT

The author would like to thank Richard Hopkins and Ryszard Janicki for their comments on draft versions of this report. The research reported in this paper was supported by a Grant from the Science and Engineering Research Council of Great Britain.

REFERENCES


APPENDIX

In this Appendix we will prove Lemma 4.5. Before that we formulate two auxiliary lemmas, where $P \in \Pi_o$ and $Q \in \Delta$. Below, if a string $s$ is a prefix of a string $t$, then we denote this by $s \preceq t$.

**Lemma A.1**

Every $w \in VFS$ can be represented as $w = w_1d_1 \ldots w_kd_kw_{k+1}$, where $(d_1, \ldots, d_k)$ is the $P$-decomposition of $w_P$ and $Ev(w_1 \ldots w_{k+1}) \subseteq Ev_Q$. 
Remark:
If $k \geq 1$ and $d_k \in \text{Cyc}(P)$ then $w_k + 1 = \varepsilon$ (follows from Proposition 3.4 and Proposition 4.3(4)).

Proof
If $k = 0$ then the lemma is trivially satisfied. Suppose $k \geq 1$.

We will show by induction that for $l = 0, \ldots, k$, $w$ can be represented as $w = u_t \bar{d}_1 \ldots u_t \bar{d}_i u_{t+1} \ldots u_{t+i} + 1$, where $Ev(u_t \ldots u_{t+i}) \subseteq EvQ$ and $Ev(u_t + 1) \subseteq Ev$. For $l = 0$ this is obviously true, so we assume that the thesis hold for $0 \leq l < k$.

Let $h_l = \bar{d}_l + 1$, for all $i \leq n$, and $\lambda = \text{first}(h_l)$ (by Proposition 4.3(2) $h_l \neq \varepsilon$).

We can represent $w$ as $w = u_1 \bar{d}_1 \ldots u_t \bar{d}_i u_{t+1} + 1 \lambda u_i$, where $\#(u_{t+1} + 1) = 0$. From $\#(u_t + 1) = 0$, Proposition 4.3(3) and $Ev(u_t \ldots u_{t+i}) \subseteq EvQ$ it follows that $Ev(u_t + 1) \subseteq EvQ$ and $h_t \not\epsilon ((\lambda \bar{u})|\bar{p})$, for all $i \leq n$.

Suppose that $h_i$ is not a prefix of $(\lambda \bar{u})|\bar{p}$. From $\lambda = \text{first}(h_i)$ and $h_i \not\epsilon ((\lambda \bar{u})|\bar{p})$, it follows that $(\lambda \bar{u})|\bar{p} = \lambda \bar{t} \bar{u}$, where $\lambda \bar{t}$ is a proper prefix of $h_i$ and $\theta \epsilon EvQ$. By Proposition 3.4, there is a $P_{(\bar{t})}$-decomposition $(\mu_1, \ldots, \mu_s)$ of $(\lambda \bar{u})|\bar{p}$, such that $\bar{d}_i \lambda \bar{t} = \mu_1 \ldots \mu_s$, for some $s \leq r$. Clearly, since $\lambda \bar{t}$ is a proper non-empty prefix of $h_i$ we must have $(\mu_1, \ldots, \mu_r) = (\bar{d}_i, \ldots, \bar{d}_i)$, contradicting Proposition 4.3(4). Hence $h_i \not\epsilon ((\lambda \bar{u})|\bar{p})$, for all $i \leq n$. This and $d_t + 1 \epsilon Ev_{(\bar{t})}$ yields $d_{t+1} \not\epsilon \lambda u$.

Thus $w$ can be represented as $w = u_t \bar{d}_1 \ldots u_t \bar{d}_i u_{t+1} + 1 \bar{d}_i u_{t+i} + 1$, where $Ev(u_{t+i} + 1) \subseteq EvQ$.

Consequently, $w$ can be represented as $w = u_t \bar{d}_1 \ldots u_t \bar{d}_i u_{t+i} + 1$. Furthermore, by $w|p = d_t \ldots d_k$, $Ev(u_t \ldots u_{tk+1}) \subseteq EvQ$, which completes the proof. \(\square\)

Lemma A.2
If $w \in VFS$ is such that $w|p$ has a proper $P$-decomposition then $w \in CVFS$.

Proof
Let $\mu = (d_1, \ldots, d_k)$ be the proper decomposition of $w|p$. By Lemma A.1 and Corollary 4.4(1), $w = w_1 \bar{d}_1 \ldots w_k \bar{d}_k u_{k+1}$, where $Ev(w_1 \ldots w_k + 1) \subseteq EvQ$.

Let $w = w_1 \bar{d}_1 \ldots w_k \bar{d}_k + 1$. Since $\mu$ is proper, in order to show the thesis it suffices to prove that $w \in VFS$. Clearly, $w \not\epsilon Ev_{(\bar{u})}$ and $d_i \not\epsilon FS(Q_{(\bar{u})})$, for all $i > n$. Hence we only need to show that $d_i \not\epsilon FS(Q_{(\bar{u})})$, for all $i \leq n$.

Let $i \leq n$. By Proposition 3.4, $w_i = t_i h_{t_i} \ldots t_i h_{t_i} + 1$, for some $t = t_1 \ldots t_i \epsilon FS(Q_{(\bar{u})})$ and a $P_{(\bar{t})}$-decomposition $(h_1, \ldots, h_t)$ of $(w|p)_t$. Thus, by Corollary 4.4(2), $l = k$ and $(h_1, \ldots, h_t) = (d_1, \ldots, d_k)$.

Consequently, $t_i h_{t_i} \ldots t_i h_{t_i} + 1 = w_i = w_1|\bar{h}_1 \ldots w_k|\bar{h}_k u_{k+1} + 1$.

By Proposition 4.2 and $h_k = d_k \not\epsilon \text{Cyc}(P_{(\bar{u})})$, $t = w_1|\bar{h}_1 \ldots, t_k + 1 = w_k + 1$.

Thus $d_i \not\epsilon FS(Q_{(\bar{u})})$, which completes the proof. \(\square\)

Proof of Lemma 4.5
(1) $VFS \subseteq P_{(\bar{w})}(CVFS)$

Let $w \in VFS$ and $(d_1, \ldots, d_k)$ be the decomposition of $w|p$.

If $k = 0$ then, by Lemma A.2, $w \not\epsilon CVFS$.

Suppose $k \geq 1$. By Proposition 4.3(5), there is $d_k \in VC_{(\bar{u})}$ such that for all $i \leq n$, if $d_i \not\epsilon \text{Cyc}(P_{(\bar{u})})$ then $d_i = \varepsilon$. We now observe that, by Remark A.1, $ww \not\epsilon VFS$. Clearly, $(d_1, \ldots, d_{k-1}, d_k)$ is the decomposition of $(ww)_p$.

Hence, by Lemma A.2, $ww \not\epsilon CVFS$, which completes the proof of (1).
(2) $υ|υ|Q ≺ υ|υ|Q$, for all $υ, υ ∈ CVFS$ such that $υ ≺ υ$.

Let $(d_1, ..., d_k)$ and $(h_1, ..., h_m)$ be the decompositions of $υ|υ|P$ and $υ|υ|P$, respectively. Furthermore, let $υ ∈ υ|υ|Q$ and $τ ∈ υ|υ|Q$. We will show that $υ ≺ τ$.

From Corollary 4.4(1) and Definition 3.6 it follows that $υ = υ_1 d_1 ... υ_k d_k + 1$ and $υ = τ_1 h_1 ... τ_m h_m + 1$, where $υ = υ_1 d_1 + ... + υ_k d_k + 1$ and $τ = τ_1 h_1 + ... + τ_m h_m + 1$.

Clearly, $υ_i ≺ τ_i$, for all $i > n$. Hence to show $υ ≺ τ$, it suffices to prove that $υ_i ≺ τ_i$, for all $i ≤ n$.

Let $i ≤ n$. By $υ ≺ τ$, $υ_1 d_1 ... υ_i d_i υ_i + 1 ≺ τ_1 h_1 ... τ_i h_i τ_i + 1$.

Suppose that $k = 0$. By Proposition 4.3(2), $h_i + 1 = ε$. Hence $υ_i ≺ τ_i ≺ τ_i ≺ τ_i$.

Suppose that $k ≥ 1$. By Corollary 4.4(3), $k ≤ m$ and $(d_1, ..., d_k) = (h_1, ..., h_k, ε)$, where $ε = h ≺ h_k$.

Clearly, by Proposition 4.3(2), $h_i ≺ ε$. Moreover, again by Proposition 4.3(2), $h_i ≺ ε$, for $j = 1, ..., m$. This and

$υ_i d_i h_i + 1 = τ_i h_i + 1 = τ_i + 1 ≺ τ_i + 1$.

yields $υ_i ≺ τ_i = τ_i ≺ τ_i$ and $υ_i + 1 ≺ τ_i + 1$. Hence

$υ_i ≺ τ_i ≺ τ_i = τ_i h_i + 1 ≺ τ_i h_i ... τ_i h_i + 1 = τ$.

This completes the proof. □