# Restructuring operations for data-flow diagrams

by Ming-Jie Chen and Chyan-Goei Chung

**When defining and designing software with structured analysis and design methods, we need to restructure data-flow diagrams. Using basic editing operations to restructure large systems with voluminous data-flow diagrams is tedious, laborious and error-prone. It is necessary to have data-flow diagram editors that provide editing operations specific for restructuring. This paper proposes and formally specifies a set of operations sufficient for all restructuring needs. It also confirms that the properties of consistency and completeness are observed by all the restructuring operations, and that both models of data-flow diagrams are equivalent, before and after each of the restructuring operations.**

## 1 Introduction

To increase engineer productivity and improve software quality, the software engineering community have made considerable research efforts towards creating better methods and tools. *Structured analysis and design* methods [1–4] are widely used in defining and designing software. Among structured techniques, *data-flow diagrams* have been reported to be the most popular [5] and contribute the most favorably towards increasing productivity when *computer-aided software engineering* (CASE) technology is used in preference to manual methods [6]. Not surprisingly, many CASE tools have been developed to help draw data-flow diagrams. Most of them support consistency analysis [7–10] and various complex drafting functions [8–11], but none of them provide a set of editing operations sufficient for conveniently restructuring data-flow diagrams. In this paper, we investigate this problem and propose a set of editing operations sufficient for all restructuring needs.

### 1.1 *The need for restructuring*

To accommodate large systems, structured analysis does not model a system in a single data-flow diagram as large as a football field, for example, but instead as *multi-levelled* data-flow diagrams. These diagrams form a strictly hierarchical structure, with *composite processes* defined in higher levels and their component processes defined in lower levels. At the top of the hierarchy is a single diagram called the *context diagram*, which contains only one process. The system is considered as partitioned from the process into all lower level diagrams, which we call *transformation diagrams*. The levelled structure makes a system model easy to read and comprehend. However, the concept of levelling generates the necessity for restructuring. The main needs are listed below.

- *Restructuring for appropriate partitioning.* Levelled data-flow diagrams allow a top-down approach to analysis, which helps us build a system model of data-flow diagrams systematically in a top-down fashion. However, the approach does not guarantee that the resulting model is an appropriate partitioning. For example, the name of a composite process may not accurately reflect everything indicated by its name in the child diagram of the process; we have to break it apart, or distribute all or part of its work to other processes. Another candidate for restructuring is a data-flow diagram that turns out to consist of disconnected networks.
- *Restructuring to reduce complexity.* We model a system as a levelled structure for readability. However, when many processes crowd a data-flow diagram in a system model, such readability is hindered; we have to split the diagram. Another important problem is interface complexity. It is a requirement that every data-flow diagram should have a fairly simple flow pattern. When the total number of flows in a diagram or the number of flows connected with a process is too large, we have to group some of the processes in the diagram together to push some flows down to a lower level.
- *Restructuring to present particular system aspects.* Most errors found during testing and operation are traceable back to poor understanding or misinterpretation of users' requirements [12]. Effective communication with users is therefore important. It is helpful to present the system model in various aspects, by restructuring the upper levels of the system model according to users' interest. Useful aspects are grouping related responses, grouping processes whose inputs and outputs are connected to a terminator or related terminators, and grouping processes to declare the interface between man and machine.
- *Restructuring for function allocation.* After completing the requirements phase, and before applying transform and transaction analyses [3, 4] to obtain structure charts from

levelled data-flow diagrams, we have to restructure them around candidate processors (if multi-processor architecture is used) and then restructure those allocated to an individual processor around candidate tasks (if multi-tasking software architecture is used for the processor). Of course, processor modelling and task modelling do more than restructuring of levelled data-flow diagrams. Nevertheless, except for the identification of processors and tasks, function allocation through restructuring is the earliest stage of the design phase.

None of the above-mentioned restructuring needs are our invention. They are found in various works by DeMarco [1], Ward and Mellor [13, 14], Hatley and Pirbhai [15], and Yourdon [16], where many convincing cases requiring restructuring can be found. We only summarise the needs here for the convenience of readers.

### *1.2 Requirements of restructuring operations*

Certainly, we can use basic editing operations, such as inserting/deleting elements, connecting/disconnecting elements, and creating/removing diagrams, to accomplish restructuring. However, this approach is tedious, laborious and error-prone. Problems of inconsistency and incompleteness may creep in. It is not easy to keep levelled data-flow diagrams in balance after restructuring. It is more difficult to guarantee that both system models are equivalent, before and after restructuring. These problems become serious for large systems with voluminous data-flow diagrams. Therefore, we need data-flow diagrams editors that provide specific operations for restructuring. An eligible set of restructuring operations must meet the following three requirements.

□ *Both system models of levelled data-flow diagrams must be equivalent, before and after every restructuring operation in the set.* The structuring of a system model into levelled diagrams is for ease of reading and comprehension. Composite processes in upper levels are only bookkeeping entities, representing ways of keeping track of connected sets of lower level processes. Thus, any restructuring operation must not change the underlying network of primitive processes.

□ *Every restructuring operation in the set must maintain the system model properties of consistency and completeness.* Restructuring is performed in those phases after a system model is completed and passes consistency and completeness analysis. Thus, the system model to be restructured is consistent and complete. Any restructuring operation must not break down the properties.

□ *The set of restructuring operations must meet the restructuring needs stated earlier.* As restructuring operations are used to automate laborious manual operations, any operation which meets some of the restructuring needs must be provided by one, or a simple combination, of the operations in the set. However, for users' convenience, it is not necessary to reduce them to a minimal set.

We have created such a set of restructuring operations. They have been formally specified and validated to meet all requirements. The formal specification and the validation are represented later in this paper.

## 2 Specification of restructuring operations

In this Section, we propose an eligible set of restructuring operations, define a formal notation based on the set, relation, and first-order logic theories [17], and then use the notation to specify the operations.

### *2.1 An eligible set of restructuring operations*

To meet all the restructuring needs mentioned above, we must have editing operations for grouping processes into composite processes (*folding*), for replacing composite processes with their component processes (*expanding*), for splitting composite processes and splitting data-flow diagrams (*splitting*), for merging composite processes and merging data-flow diagrams (*merging*), and for moving processes from their residing data-flow diagrams to other diagrams (*transferring*). Therefore, we have at least five restructuring operations, including *folding*, *expanding*, *splitting*, *merging* and *transferring*. The five operations are sufficient for all the restructuring needs and are discussed below. A schematic illustration is shown in Fig. 1.

- *Folding:* groups a set of processes residing in the same data-flow diagram into a composite process, creates a child diagram for the composite process, and puts the set of processes, data flows and data stores linked with these processes, and their connections into the child diagram. This operation increases a system model of levelled data-flow diagrams by one level.
- *Expanding:* replaces a composite process in its residing diagram, with all the elements and their interconnected network in its child diagram. This operation decreases a system model by one level.
- *Splitting:* splits a composite process and its child diagram into two composite processes and two diagrams, with each new composite process being either of the two new diagrams.
- *Merging:* merges two composite processes in the same diagram into one composite process, merges their child diagram into one diagram, and makes the new diagram become the child diagram of the new composite process.
- *Transferring:* transfers a process from its current residing diagram into another. Of course, we cannot transfer the only process in the context diagram to another diagram; we do not allow another process to be transferred to the context diagram. Besides, we cannot transfer any process to any diagram decomposed directly or indirectly from the process; otherwise, cyclic decomposition sequences will be introduced into the system model.

### *2.2 The notation*

We shall specify each operation as an editing operator. Before the specification, we describe below the notation used, including a formal definition of the structures of data-flow diagrams and system models of levelled data-flow diagrams, and a formula for expressing restructuring operators.

*2.2.1 Data-flow diagram structures:* following DeMarco's structured analysis [1], we use four kinds of elements (i.e. processes, terminators, flows, and stores) and connections

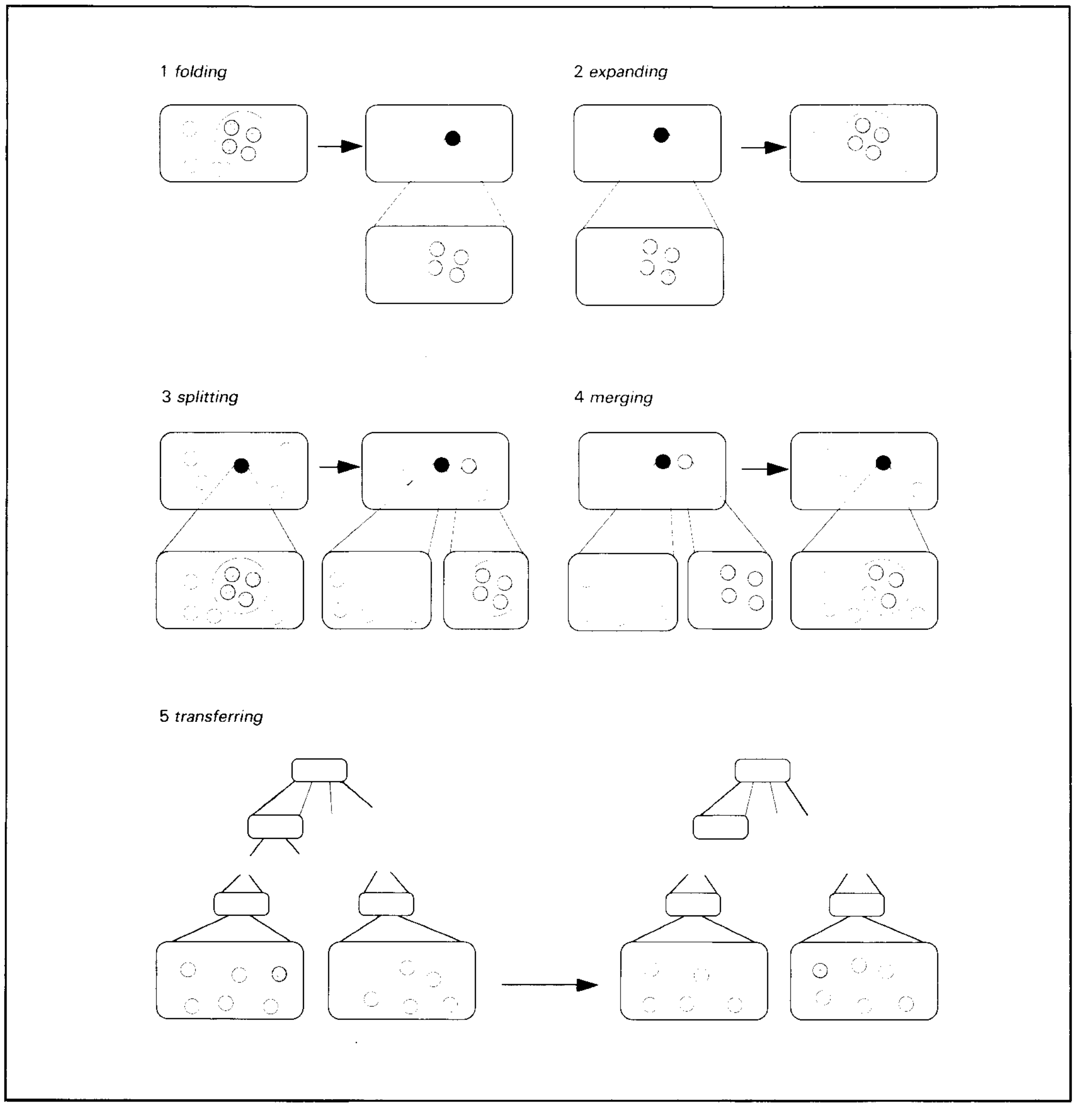


**Fig. 1 Schematic illustration of restructuring operations**

between the elements to build a data-flow diagram. Thus, we have

*Definition 1*

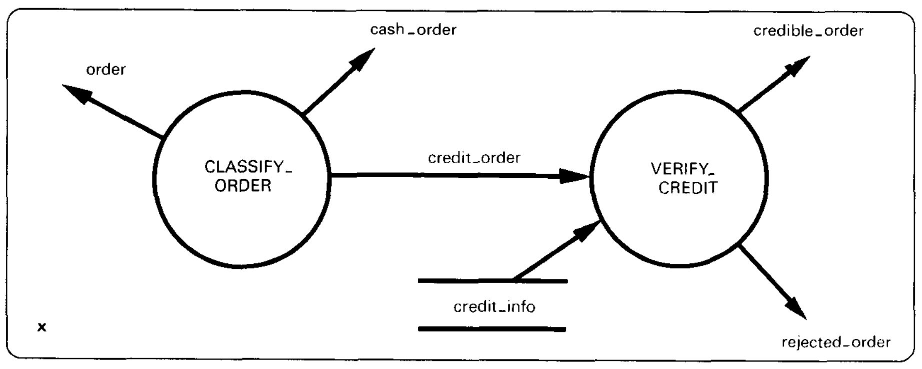
A *data-flow diagram* (*structure*) is a five-tuple ($\boldsymbol{P}$, $\boldsymbol{T}$ $\boldsymbol{F}$, $\boldsymbol{S}$, $\boldsymbol{C}$), where $P$, $T$, $F$ and $S$ are sets of *processes*, *terminators*, *flows* and *stores*, respectively, and $C \subseteq [(P \cup T) \times (F \cup S)] \cup [(F \cup S) \times (P \cup T)]$ is a *connection relation*.

An initially created data-flow diagram will be ($\varnothing$, $\varnothing$, $\varnothing$, $\varnothing$, $\varnothing$). In this definition, we assume that every element is given a label when created and it will be referred to by its label afterwards. For simplicity, we do not introduce a labelling function for the definition. When we talk about the equivalence of two processes, such as $p1 = p2$ and $p1 \neq p2$, we mean whether they have the same label, such as *Label*(p1) = *Label*(p2) and *Label*(p1) ≠ *Label*(p2). Similar conventions are applied to terminators, flows and stores.

For a data-flow diagram structure $x$, we use the notations $\boldsymbol{x.P}$, $\boldsymbol{x.T}$, $\boldsymbol{x.F}$, $\boldsymbol{x.S}$ and $\boldsymbol{x.C}$ to refer to its five components. To shorten expressions, we define $\boldsymbol{X.Y}$ as $\bigcup_{x \in X} x.Y$, where $X$ is a set of data-flow diagrams and $Y$ can be $P$, $T$, $F$, $S$ or $C$. We also define four functions for obtaining the flow inputs, flow outputs, store inputs and store outputs of processes and terminators.

*Definition 2*

For a process or terminator $e$, in its residing data-flow diagram $x$, and for a set of processes and terminators $E$, a *flow input function* $\boldsymbol{F^i}$(_), a *flow output function* $\boldsymbol{F^o}$(_), a *store input function* $\boldsymbol{S^i}$(_) and a *store output function* $\boldsymbol{S^o}$(_) are defined as follows:



*b*

$x = (P, T, F, S, C)$
$P = \{$CLASSIFY_ORDER, VERIFY_CREDIT$\}$
$T = \varnothing$
$F = \{$order, cash_order, credit_order, credible_order, rejected_order$\}$
$S = \{$credit_info$\}$
$C = \{\langle$order, CLASSIFY_ORDER$\rangle$, $\langle$CLASSIFY_ORDER, cash_order$\rangle$, $\langle$CLASSIFY_ORDER, credit_order$\rangle$, $\langle$credit_order, VERIFY_CREDIT$\rangle$, $\langle$credit_info, VERIFY_CREDIT$\rangle$, $\langle$VERIFY_CREDIT, credibile_order$\rangle$, $\langle$VERIFY_CREDIT, rejected_order$\rangle\}$

*c*

$F^i$(CLASSIFY_ORDER) $= \{$order$\}$
$F^o$(CLASSIFY_ORDER) $= \{$cash_order, credit_order$\}$
$S^i$(CLASSIFY_ORDER) $= \varnothing$
$S^o$(CLASSIFY_ORDER $= \varnothing$
$F^i$(VERIFY_CREDIT) $= \{$credit_order$\}$
$F^o$(VERIFY_CREDIT) $= \{$credible_order, rejected_order$\}$
$S^i$(VERIFY_CREDIT) $= \{$credit_info$\}$
$S^o$(VERIFY_CREDIT) $= \varnothing$

**Fig. 2 Illustration of the data-flow diagram structure and the flow/store input/output functions**
*a* A sample transformation diagram *b* Definition of the diagram's structure *c* The flow/store input/output functions

$$F^i(e) = \{f \in x.F \mid \langle f, e\rangle \in x.C \text{ for some } e \in x.P \cup x.T\}$$
$$F^o(e) = \{f \in x.F \mid \langle e, f\rangle \in x.C \text{ for some } e \in x.P \cup x.T\}$$
$$S^i(e) = \{s \in x.S \mid \langle s, e\rangle \in x.C \text{ for some } e \in x.P \cup x.T\}$$
$$S^o(e) = \{s \in x.S \mid \langle e, s\rangle \in x.C \text{ for some } e \in x.P \cup x.T\}$$
$$F^i(E) = \bigcup_{e \in E} F^i(e)$$
$$F^o(E) = \bigcup_{e \in E} F^o(e)$$
$$S^i(E) = \bigcup_{e \in E} S^i(e)$$
$$S^o(E) = \bigcup_{e \in E} S^o(e)$$

Context diagrams and transformation diagrams are special data-flow diagrams. The same structural format as the data-flow diagram structure is used to represent context diagrams and transformation diagrams. A sample transformation diagram is given in Fig. 2 to familiarise readers with the notations and functions.

*2.2.2 System models:* We model a system as a levelled structure of data-flow diagrams, where the top one is a context diagram and all lower level diagrams are transformation diagrams. Thus, we can define a system model as a context diagram, a set of transformation diagrams, and a mapping between these transformation diagrams and their parent processes.

*Definition 3*

A *system model* is a three-tuple $(\boldsymbol{cd}, \boldsymbol{X}, \boldsymbol{D})$, where *cd* is a *context diagram*, $X$ is a set of *transformation diagrams* and $D \subseteq \mathbb{P} \times X$ is a *decomposition relation*, with $\mathbb{P} = cd.P \cup X.P$.

An initially created system model will be $((\varnothing, \varnothing, \varnothing, \varnothing, \varnothing), \varnothing, \varnothing)$. For a system model *m*, we use the notations $\boldsymbol{m.cd}$, $\boldsymbol{m.X}$ and $\boldsymbol{m.D}$ to refer to its three components, and the notations $\boldsymbol{m.P}$, $\boldsymbol{m.T}$, $\boldsymbol{m.F}$, $\boldsymbol{m.S}$ and $\boldsymbol{m.C}$ to refer to the sets of all processes, terminators, flows, stores, and their connections in it, i.e. $m.P = m.cd.P \cup m.X.P$, $m.T = m.cd.T$, $m.F = m.cd.F \cup m.X.F$, $m.S = m.X.S$ and $m.C = m.cd.C \cup m.X.C$.

For every $\langle p, x\rangle$ in $m.D$, we use the function $\boldsymbol{x.pa}$ to refer to the parent process $p$ of the diagram $x$, and the function ***p.kid*** to refer to the child diagram $x$ of the process $p$. If $p$ does not exist in $m.P$ such that $\langle p, x\rangle$ is in $m.D$, the function $x.pa$ is undefined. If $x$ does not exist in $m.X$ such that $\langle p, x\rangle$ is in $m.D$, the function $p.kid$ is undefined.

*2.2.3 Formula for expressing restructuring operators:* before restructuring, we must make sure that such a process is allowed. Thus, a formula for expressing restructuring operators consists of a prerequisite and some editing actions. An operator is specified as

```
operator-name(parameter1, parameter2, ...) = =
require a-prerequisite
perform
  editing-action-1
  editing-action-2
  ...
end
```

An editing action changes the structure of a system model or one of the data-flow diagram structures in the model. We use the notation \ to specify the structure to change and the structure after change, and use the key word **where** to indicate a newly created data-flow diagram structure if required. An editing action is specified as

*the-structure-to-change* \ *the-structure-after-change*,
[**where** *specification-of-a-newly-created data-flow-diagram-structure*];

We also use the keyword **then** to force a list of editing actions to be taken one after another. For simplicity we allow recursive definition, and for flexibility we allow an action to be conditional and existentially or universally quantified, such as

*condition* ⇒ *an-action*
$(\exists x)$(*condition* ⇒ *an-action*)
$(\forall x)$(*condition* ⇒ *an-action*)

## 2.3 Specification of the restructuring operators

Now, we specify the five restructuring operators with the notations. To ease understanding, naming of parameters and variables in the specification is shown in Fig. 3. The specification only shows what changes each of the operators will make to the addressed system model; no implementations are indicated. The expression for each of the $F_i$s and $S_i$s in the specification can be reformulated in many other ways. For brevity, for a process (or a set of processes) terminators and all other processes that are not in the diagrams directly or indirectly decomposed from the process (the set of processes) are called the *externals* of the process (the set of processes); the flows and stores that are connected both with the process (the set of processes) and with the externals of the process (the set of processes) are called the *interfacing flows and stores* of the process (the set of processes); and the connections between the interfacing

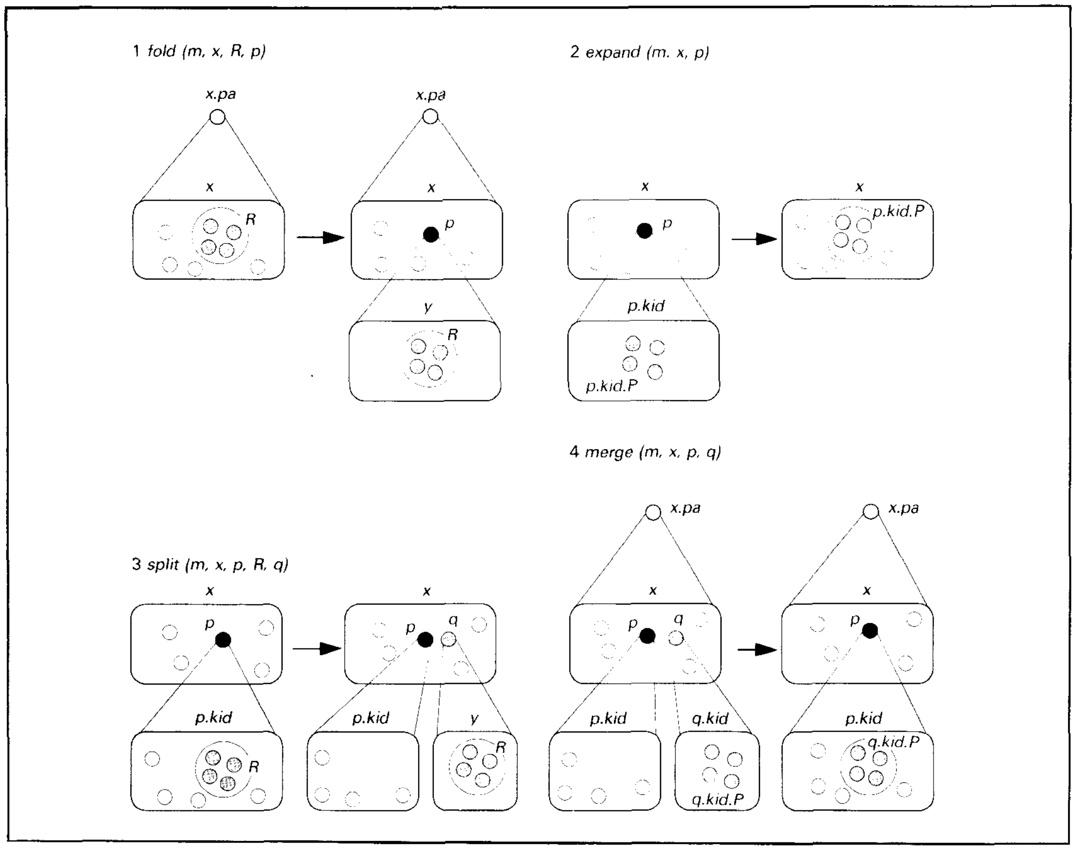


**Fig. 3 Naming of parameters and variables in the specification of restructuring operators**

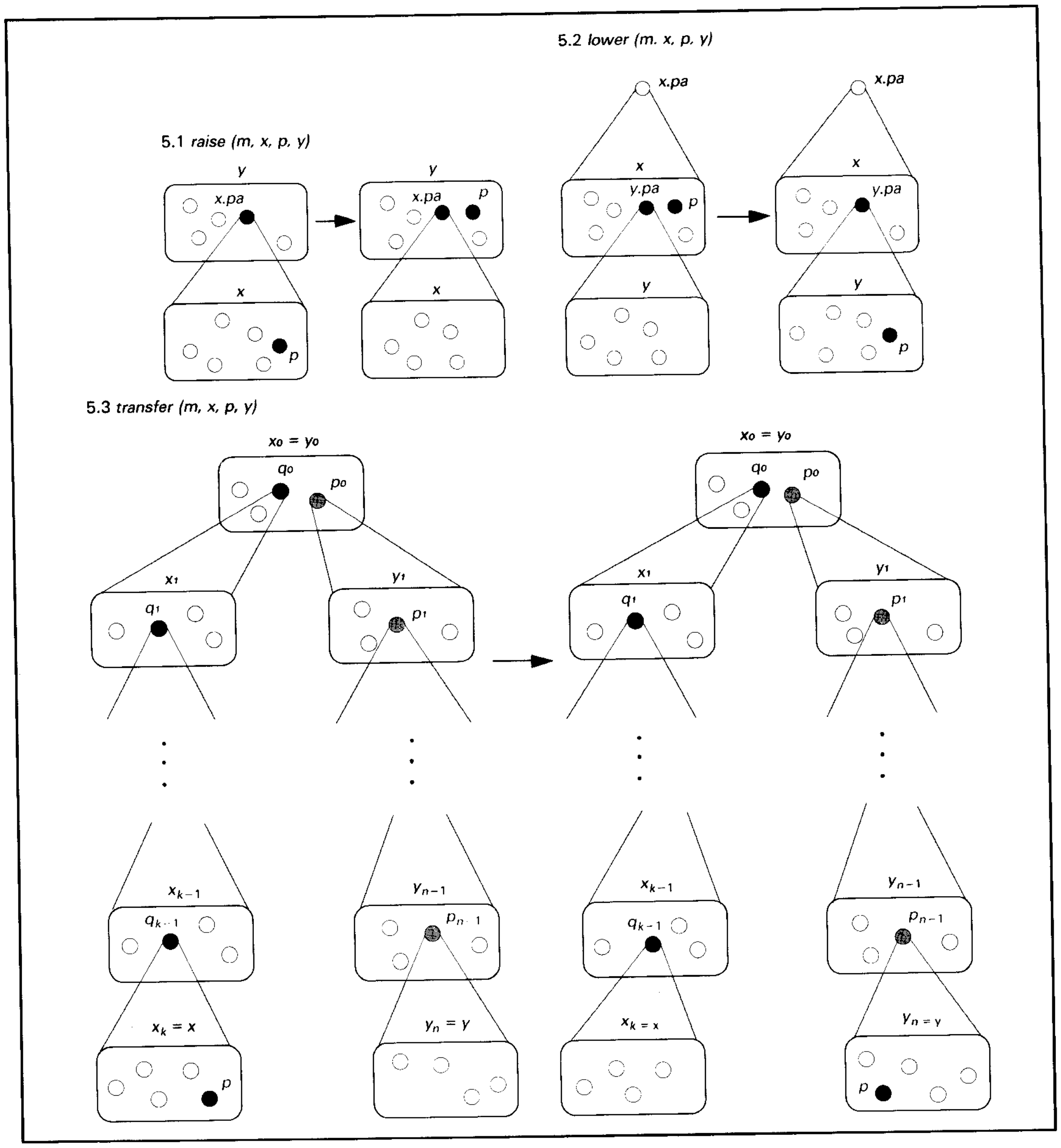


**Fig. 3 (continued) Naming of parameters and variables in the specification of restructuring operators**

flows and stores and the process (the set of processes) are called the *interfacing connections* with the process (the set of processes). In addition, note that we inhibit any restructuring operator from making changes to the context diagram in a system model.

*2.3.1 The fold operator:* it groups a set of processes ($R$), in a transformation diagram ($x$) in a system model ($m$), into a composite process ($p$). The flows and stores that are only connected internally with the processes in the set are all removed from the diagram. All the connections with the processes are also removed, but the original interfacing connections with the set of processes become the interfacing connections with the composite process. The operator also creates a diagram ($y$) as the child diagram of the process ($p$) for accommodating the set of processes ($R$) and their original connected flows, connected stores and connections.

```
**fold**$(m, x, R, p) = =$
**require** $(x \in m.X) \wedge (R \subset x.P) \wedge (R \neq \varnothing)$
                                   $\wedge (R \neq x.P) \wedge (p \notin m.P)$
**perform**
  $x \setminus (x.P \cup \{p\} - R, x.T, x.F - F3, x.S - S3,$
    $x.C \cup [(F1 \cup S1) \times \{p\}] \cup [\{p\}$
                                   $\times (F2 \cup S2)] - C1)$
  $m \setminus (m.cd, m.X \cup \{y\}, m.D \cup \{\langle p, y\rangle\}),$
                         **where** $y = (R, \varnothing, F4, S4, C1)$
**end**
```

where

$F1 = F^{i}(R) - F^{o}(R)$

$F2 = F^o(R) - F^i(R)$
$F3 = F^i(R) \cap F^o(R)$
$F4 = F^i(R) \cup F^o(R)$
$S1 = S^i(R) \cap [S^i(x.P - R) \cup S^o(x.P - R) \cup S^i(x.pa)]$
$S2 = S^o(R) \cap [S^i(x.P - R) \cup S^o(x.P - R) \cup S^o(x.pa)]$
$S3 = [S^i(R) \cup S^o(R)] - [S^i(x.P - R) \cup S^o(x.P - R) \cup S^i(x.pa) \cup S^o(x.pa)]$
$S4 = S^i(R) \cup S^o(R)$
$C1 = \bigcup_{r \in R}[(F^i(r) \cup S^i(r)) \times \{r\}] \cup [\{r\} \times (F^o(r) \cup S^o(r))]$

*2.3.2 The expand operator:* it replaces a composite process ($p$), in its residing diagram ($x$) in a system model ($m$), with the contents of its child diagram ($p.kid$). The interfacing connections with the process are removed from its residing diagram. The process' child diagram and the decomposition relationship between the process and its child diagram are removed from the system model.

```
expand(m, x, p) = =
require $(x \in m.X) \wedge (p \in x.P) \wedge (\exists z \in m.X)(\langle p, z\rangle \in m.D)$
perform
    $x \setminus (x.P \cup p.kid.P - \{p\}, x.T \cup p.kid.T, x.F \cup F1, x.S \cup S1,$
      $x.C \cup p.kid.C - C1)$
        $m \setminus (m.cd, m.X - \{p.kid\}, m.D - \{\langle p, p.kid\rangle\})$
end
```

where

$F1 = p.kid.F - [F^i(p) \cup F^o(p)]$
$S = p.kid.S - [S^i(p) \cup S^o(p)]$
$C1 = [(F^i(p) \cup S^i(p)) \times \{p\}] \cup [\{p\} \times (F^o(p) \cup S^o(p))]$

*2.3.3 The split operator:* it splits a composite process ($p$), in a transformation diagram ($x$) in a system model ($m$), into two composite processes ($p$ and $q$) by separating a set of processes ($R$) in its child diagram ($p.kid$) into a newly created diagram ($y$). The flows and stores that are connected both with the set of processes and with the other processes in the child diagram ($p.kid$), but not with the composite process ($p$), are added to the transformation diagram. The interfacing connections with the set of processes become the interfacing connections with the newly created composite process. The interfacing connections with the other processes in the child diagram are added to the interfacing connections with the original composite process, if they were not originally with it. However, the interfacing connections that are with the original composite process but not with the other processes in the child diagram, other than the set of processes, are removed from the transformation diagram. The newly created diagram becomes the child diagram of the newly created composite process ($q$). The operator adds all the connected flows, connected stores and connections with the set of processes to the newly created diagram.

```
split(m, x, p, R, q) = =
require $(x \in m.X) \wedge (p \in x.P) \wedge (\exists z \in m.X)(\langle p, z\rangle \in m.D) \wedge (R \subset p.kid.P)$
  $\wedge (R \neq \emptyset) \wedge (R \neq p.kid.P) \wedge (q \notin m.P)$
perform
    $x \setminus (x.P \cup \{q\}, x.T, x.F \cup F1 \cup F2, x.S \cup (S1 \cup S2),$
      $x.C \cup [(F3 \cup S3) \times \{q\}] \cup [\{q\} \times (F4 \cup S4)]$
      $\cup [(F1 \cup S1) \times \{p\}]$
      $\cup [\{p\} \times (F2 \cup S2)] - [(F5 \cup S5) \times \{p\}]$
      $- [\{p\} \times (F6 \cup S6)])$
    $p.kid \setminus (p.kid.P - R, p.kid.T, p.kid.F - F7,$
      $p.kid.S - S7, p.kid.C - C1)$
    $m \setminus (m.cd, m.X \cup \{y\}, m.D \cup \{\langle q, y\rangle\}),$
      where $y = (R, \emptyset, F8, S8, C1)$
end
```

where

$F1 = F^i(p.kid.P - R) \cap F^o(R)$
$F2 = F^o(p.kid.P - R) \cap F^i(R)$
$F3 = F^i(R) - F^o(R)$
$F4 = F^o(R) - F^i(R)$
$F5 = F^i(p) \cap F^i(R)$
$F6 = F^o(p) \cap F^o(R)$
$F7 = [F^i(R) \cup F^o(R)] - F1 - F2$
$F8 = F^i(R) \cup F^o(R)$
$S1 = [S^i(p.kid.P - R) - S^i(p)] \cap [S^i(R) \cup S^o(R)]$
$S2 = [S^o(p.kid.P - R) - S^o(p)] \cap [S^i(R) \cup S^o(R)]$
$S3 = S^i(R) \cap [S^i(p.kid.P - R) \cup S^o(p.kid.P - R) \cup S^i(p)]$
$S4 = S^o(R) \cap [S^i(p.kid.P - R) \cup S^o(p.kid.P - R) \cup S^o(p)]$
$S5 = S^i(p) - S^i(p.kid.P - R)$
$S6 = S^o(p) - S^o(p.kid.P - R)$
$S7 = [S^i(R) \cup S^o(R)] - [S^i(p.kid.P - R) \cup S^o(p.kid.P - R)]$
$S8 = S^i(R) \cup S^o(R)$
$C1 = \bigcup_{r \in R}[(F^i(r) \cup S^i(r)) \times \{r\}] \cup [\{r\} \times (F^o(r) \cup S^o(r))]$

*2.3.4 The merge operator:* it merges two composite processes ($p$ and $q$), in the same diagram ($x$) in a system model ($m$), into one composite process ($p$). The flows and stores that are only connected internally with the two processes, and the connections between the remaining process ($p$) and these flows and stores are all removed from the diagram. All the connections with the removed process ($q$) are also removed, but those connections that are between the removed process and the externals of the two composite processes are added to the interfacing connections with the remaining process, if they were not originally with it. The operator also merges the child diagrams of the two composite processes ($p.kid$ and $q.kid$) into one diagram ($p.kid$). The other diagram ($q.kid$) and its decomposition relationship with the removed process are both removed from the system model.

```
merge(m, x, p, q) = =
require $(x \in m.X) \wedge (p, q \in x.P) \wedge (p \neq q)$
  $\wedge (\exists y, z \in m.X)(\langle p, y\rangle, \langle q, z\rangle \in m.D)$
perform
    $x \setminus (x.P - \{q\}, x.T, x.F - F1 - F2, x.S$
      $- (S1 \cup S2), x.C \cup [(F3 \cup S3) \times \{p\}]$
      $\cup [\{p\} \times (F4 \cup S4)] - C1 - [(F1 \cup S1) \times \{p\}] - [\{p\} \times (F2 \cup S2)])$
  $p.kid \setminus (p.kid.P \cup q.kid.P, p.kid.T \cup q.kid.T,$
      $p.kid.F \cup F5, p.kid.S \cup S5,$
      $p.kid.C \cup q.kid.C)$
    $m \setminus (m.cd, m.X - \{q.kid\}, m.D - \{\langle q, q.kid\rangle\})$
end
```

where

$F1 = F^i(p) \cap F^o(q)$
$F2 = F^o(p) \cap F^i(q)$

$$
\begin{aligned}
F3 &= F^i(q) - F^o(p)\\
F4 &= F^o(q) - F^i(p)\\
F5 &= q.kid.F - F1 - F2\\
S1 &= S^i(p) - [S^i(x.P - \{p, q\}) \cup S^o(x.P - \{p, q\}) \\
&\qquad \cup S^i(x.pa)]\\
S2 &= S^o(p) - [S^i(x.P - \{p, q\}) \cup S^o(x.P - \{p, q\}) \\
&\qquad \cup S^o(x.pa)]\\
S3 &= [S^i(q) - S^i(p)] \cap [S^i(x.P - \{p, q\}) \cup S^o(x.P \\
&\qquad - \{p, q\}) \cup S^i(x.pa)]\\
S4 &= [S^o(q) - S^o(p)] \cap [S^i(x.P - \{p, q\}) \cup S^o(x.P \\
&\qquad - \{p, q\}) \cup S^o(x.pa)]\\
S5 &= p.kid.S - [(S^i(p) \cup S^o(p)) \cap (S^i(q) \cup S^o(q))]\\
C1 &= [(F^i(q) \cup S^i(q)) \times \{q\}] \cup [\{q\} \times (F^o(q) \cup S^o(q))]
\end{aligned}
$$

*2.3.5 The transfer operator:* it transfers a process ($p$), from its residing diagram ($x$) in a system model ($m$), into another diagram ($y$). To specify the operator clearly, we use two more primitive operators: the *raise* operator, which moves a process ($p$) from its residing diagram ($x$) one level up to the diagram ($y$), in which its parent process ($x.pa$) resides; and the *lower* operator, which moves a process ($p$) from its residing diagram ($x$) one level down to the child diagram ($y$) of one of its sibling processes ($y.pa$). To save space, detailed descriptions of these three operators are omitted.

```
**raise**$(m, x, p, y) = =$
**require** $(x, y \in m.X) \wedge (x.pa \in y.P) \wedge (p \in x.P)$
                                              $\wedge (\{p\} \neq x.P)$
**perform**
  $y \setminus (y.P \cup \{p\}, y.T, y.F \cup F1 \cup F2, y.S \cup (S1 \cup S2)$
    $y.C \cup C1 \cup [(F1 \cup S1) \times \{x.pa\}]$
                                    $\cup [\{x.pa\} \times (F2 \cup S2)]$
      $- [(F3 \cup S3) \times \{x.pa\}] - [\{x.pa\} \times (F4$
                                    $\cup S4)])$
  $x \setminus (x.P - \{p\}, x.T, x.F - F3 - F4, x.S - S5, x.C - C1)$
**end**
```

where

$$
\begin{aligned}
F1 &= F^i(x.P - \{p\}) \cap F^o(p)\\
F2 &= F^o(x.P - \{p\}) \cap F^i(p)\\
F3 &= F^i(x.pa) \cap F^i(p)\\
F4 &= F^o(x.pa) \cap F^o(p)\\
S1 &= [S^i(x.P - \{p\}) - S^i(x.pa)] \cap [S^i(p) \cup S^o(p)]\\
S2 &= [S^o(x.P - \{p\}) - S^o(x.pa)] \cap [S^i(p) \cup S^o(p)]\\
S3 &= S^i(x.pa) - S^i(x.P - \{p\})\\
S4 &= S^o(x.pa) - S^o(x.P - \{p\})\\
S5 &= [S^i(p) \cup S^o(p)] - [S^i(x.P - \{p\}) \cup S^o(x.P - \{p\})]\\
C1 &= [(F^i(p) \cup S^i(p)) \times \{p\}] \cup [\{p\} \times (F^o(p) \cup S^o(p))]
\end{aligned}
$$

```
**lower**$(m, x, p, y) = =$
**require** $(x, y \in m.X) \wedge (y.pa \in x.P) \wedge (p \in x.P)$
                                              $\wedge (p \neq y.pa)$
**perform**
  $x \setminus (x.P - \{p\}, x.T, x.F - F1 - F2, x.S - (S1 \cup S2),$
    $x.C \cup [(F3 \cup S3) \times \{y.pa\}] \cup [\{y.pa\} \times (F4 \cup S4)]$
      $- C1 - [(F1 \cup S1) \times \{y.pa\}] - [\{y.pa\} \times (F2$
                                    $\cup S2)])$
  $y \setminus (y.P \cup \{p\}, y.T, y.F \cup F3 \cup F4, y.S \cup S5, y.C \cup C1)$
**end**
```

where

$$
\begin{aligned}
F1 &= F^i(y.pa) \cap F^o(p)\\
F2 &= F^o(y.pa) \cap F^i(p)\\
F3 &= F^i(p) - \overline{F}^o(y.pa)\\
F4 &= F^o(p) - F^i(y.pa)\\
S1 &= S^i(y.pa) - [S^i(x.S - \{y.pa, p\}) \\
&\qquad \cup S^o(x.S - \{y.pa, p\}) \cup S^i(x.pa)]\\
S2 &= S^o(y.pa) - [S^i(x.S - \{y.pa, p\}) \\
&\qquad \cup S^o(x.S - \{y.pa, p\}) \cup S^o(x.pa)]\\
S3 &= [S^i(p) - S^i(y.pa)] \cap [S^i(x.P - \{y.pa, p\}) \\
&\qquad \cup S^o(x.P - \{y.pa, p\}) \cup S^i(x.pa)]\\
S4 &= [S^o(p) - S^o(y.pa)] \cap [S^i(x.P - \{y.pa, p\}) \\
&\qquad \cup S^o(x.P - \{y.pa, p\}) \cup S^o(x.pa)]\\
S5 &= [S^i(p) \cup S^o(p)] - [S^i(y.pa) \cup S^o(y.pa)]\\
C1 &= [(F^i(p) \cup S^i(p)) \times \{p\}] \cup [\{p\} \times (F^o(p) \cup S^o(p))]
\end{aligned}
$$

```
**transfer**$(m, x, p, y) = =$
**require** $(x, y \in m.X) \wedge (x \neq y) \wedge (p \in x.P) \wedge (\{p\} \neq x.P)$
  $\wedge \neg(\exists y_0, p_0, y_1, \ldots, p_{n-1}, y_n)[(1 \leqslant n) \wedge (p_0 = p)$
                                   $\wedge (y_0 = x) \wedge (y_n = y) \wedge$
  $(p_0 \in y_0, P) \wedge \cdots \wedge (p_{n-1} \in y_{n-1}.P) \wedge (\langle p_0, y_1 \rangle,$
                                   $\ldots, \langle p_{n-1}, y_n \rangle \in m.D)]$
**perform**
  $(\exists x_0, q_0, x_1, \ldots, q_{k-1}, x_k)[(1 \leqslant k) \wedge (x_0 = y)$
                                   $\wedge (x_k = x) \wedge (q_0 \in x_0.P)$
    $\wedge \cdots \wedge (q_{k-1} \in x_{k-1}.P) \wedge (\langle q_0, x_1 \rangle, \ldots,$
                                   $\langle q_{k-1}, x_k \rangle \in m.D)$
    $\Rightarrow$ *raise*$(m, x_k, p, x_{k-1})$ **then** $\cdots$
                                   **then** *raise*$(m, x_1, p, x_0)]$

  $(\exists y_0, p_0, y_1, \ldots, p_{n-1}, y_n)[(1 \leqslant n) \wedge (y_0 = x)$
                                   $\wedge (y_n = y) \wedge (p_0 \in y_0.P)$
    $\wedge \cdots \wedge (p_{n-1} \in y_{n-1}.P) \wedge (\langle p_0, y_1 \rangle, \ldots,$
                                   $\langle p_{n-1}, y_n \rangle \in m.D)$
    $\Rightarrow$ *lower*$(m, y_0, p, y_1)$ **then** $\cdots$
                                   **then** *lower*$(m, y_{n-1}, p, y_n)]$
  $(\exists x_0, q_0, x_1, \ldots, q_{k-1}, x_k)(\exists y_0, p_0, y_1, \ldots, p_{n-1}, y_n)$
    $[(1 \leqslant k) \wedge (1 \leqslant n) \wedge (x_0 = y_0) \wedge (q_0 \neq p_0)$
                                   $\wedge (x_k = x) \wedge (y_n = y)$
    $\wedge (q_0 \in x_0.P) \wedge \cdots \wedge (q_{k-1} \in x_{k-1}.P) \wedge (p_0 \in y_0.P)$
                                   $\wedge \cdots \wedge (p_{n-1} \in y_{n-1}.P)$
    $\wedge (\langle q_0, x_1 \rangle, \ldots, \langle q_{k-1}, x_k \rangle, \langle p_0, y_1 \rangle, \ldots,$
                                   $\langle p_{n-1}, y_n \rangle \in m.D)$
    $\Rightarrow$ *raise*$(m, x_k, p, x_{k-1})$ **then** $\cdots$ **then**
                                   *raise*$(m, x_1, p, x_0)$ **then**
      *lower*$(m, y_0, p, y_1)$ **then** $\cdots$ **then**
                                   *lower*$(m, y_{n-1}, p, y_n)]$
**end**
```

## 3 Validation of the restructuring operators

In this Section we formally define the meaning of consistency and completeness, and the equivalence of system models. We also confirm that the properties of consistency and completeness are observed by *all* the restructuring operations, and that both system models are equivalent, before and after each of the restructuring operations.

### *3.1 Definition of consistency and completeness*

The structure of a system model, context diagram and transformation diagram must be consistent and complete, and regulated by a set of conventions called *formation rules*. These rules include ways of forming a system model from data-flow diagrams, of forming a context diagram from basic elements and of forming a transformation diagram from basic elements. They also include the balancing (i.e. the equivalence of flows and stores and the consistency of their connections) between composite processes and their child transformation diagrams, the originality of elements (labels) over all the diagrams in a system model, and the accessibility of processes to the externals.

We formally define a formation rule as a *well formed formula* [17]. For example, we define the rule inhibiting any transformation diagram with more than one parent process as

$$(\forall x \in m.X)(\forall p, q \in m.P)[(\langle p, x\rangle, \langle q, x\rangle \in m.D) \Rightarrow (p = q)]$$

In this formula, the symbol $\Rightarrow$ denotes 'implies' and the letter $m$ refers to any system model. We also use set connectives to express well formed formulas concisely. For example, the rule only allowing flows and stores to be connected with processes in a transformation diagram can be defined in the following formula.

$$x.C \subseteq [(x.F \cup x.S) \times x.P] \cup [x.P \times (x.F \cup x.S)]$$

In this formula, the symol $\subseteq$ denotes 'is a subset of' and the letter $x$ refers to any transformation diagram.

A typical set of commonly used formation rules is described and formally defined in Appendix 1. When a system model structurally conforms to all of the formation rules, it is *structurally consistent and complete.*

## *3.2 Definition of equivalence of system models*

What we are really concerned about in a system model are primitive processes and their interfaces, through data flows and stores, with one another and with terminators. The structuring of a system model into levelled data-flow diagrams is only to ease understanding. In a structurally consistent and complete system model, every composite process can be completely replaced by the contents of its child diagram. We can repeatedly apply the replacement process to a system model until a single data-flow diagram, consisting only of primitive processes, terminators, flows, stores and their connections, is obtained. We call the single data-flow diagram *the intrinsic model* of the system model and define two system models as equivalent if they have the same intrinsic model.

*Definition 4*

For a structurally consistent and complete system model $m$, its *intrinsic model* is a data-flow diagram structure ($\mathbb{P}$, $m.T$, $m.F$, $m.$S, $\mathbb{C}$), where

$$\mathbb{P} = \{p \in m.P \mid \neg(\exists z \in m.X)(\langle p, z\rangle \in m.D)\}$$

$$\mathbb{C} = \bigcup_{t \in m.T}(F^i(t) \times \{t\}) \cup (\{t\} \times F^o(t)) \cup \bigcup_{p \in \mathbb{P}}[(F^i(p) \cup S^i(p)) \times \{p\}] \cup [\{p\} \times (F^o(p) \cup S^o(p))]$$

*Definition 5*

Two structurally consistent and complete system models are *equivalent* if, and only if, they have the same intrinsic model.

## *3.3 The validation*

Since the validation for all the restructuring operators is similar, we only show the validation of the *fold* operator here. Recall that the *fold* operator groups a set of processes ($R$), in a transformation diagram ($x$) in a system model ($m$), into a composite process ($p$), and it also creates a diagram ($y$) as the child diagram of the process ($p$), for accommodating the set of processes ($R$) and their connected flows and stores. We refer to the system model after the operation as $m'$ and the transformation diagram changed from $x$ as $x'$.

To confirm that the *fold* operator observes the properties of consistency and completeness is to show that the system model after folding ($m'$) conforms to all the formation rules, on the condition that the system model up to fold ($m$) conforms to all the formation rules. The conformity of the system model $m'$ to each of the formation rules is shown in Appendix 2.

To demonstrate that the two system models $m'$ and $m$ are equivalent, we must show that the five components in their intrinsic models are equivalent.

- From the specification of *fold*, we have $m'.\mathrm{cd} = m.\mathrm{cd}$, $m'.X = (m.X - \{x\}) \cup \{x', y\}$, $x'.P = x.P \cup \{p\} - R$, and $y.P = R$. Thus, we have $x'.P \cup y.P = x.P \cup \{p\}$. We then have $m'.X.P = (m.X.P - x.P) \cup (x'.P \cup y.P) = (m.X.P - x.P) \cup (x.P \cup \{p\}) = m.X.P \cup \{p\}$. Therefore, we have $m'.P = m'.cd.P \cup m'.X.P = m.cd.P \cup m.X.P \cup \{p\} = m.P \cup \{p\}$, i.e. $m'$ and $m$ have the same set of processes, except $p$. Since $\langle p, y\rangle \in m'.D$, namely, $p$ is not a primitive process, $m'$ and $m$ have the same set of primitive processes.
- As $m'.cd = m.cd$, we have $m'.T = m'.cd.T = m.cd.T = m.T$.
- Similarly, we have $x'.F \cup y.F = [x.F - (F^i(R) \cap F^o(R))] \cup [F^i(R) \cup F^o(R)] = x.F$. We then have $m'.X.F = (m.X.F - x.F) \cup (x'.F \cup y.F) = m.X.F$. Therefore, we have $m'.F = m'.cd.F \cup m'.X.F = m.cd.F \cup m.X.F. = m.F$.
- Similarly, we have $x'.S \cup y.S = [x.S - ([S^i(R) \cup S^o(R)] - [S^i(x.P - R) \cup S^o(x.P - R) \cup S^i(x.pa) \cup S^o(x.pa)])] \cup [(S^i(R) \cup S^o(R)] = x.S$. We then have $m'.X.S = (m.X.S - x.S) \cup (x'.S \cup y.S) = m.X.S$. Therefore, we have $m'.S = m'X.S - m.X.S = m.S$.
- Similarly, we have $x'.C \cup y.C = [x.C \cup [(F1 \cup S1) \times \{p\}] \cup [\{p\} \times (F2 \cup S2)] - C1] \cup C1 = x.C \cup [(F1 \cup S1) \times \{p\}] \cup [\{p\} \times (F2 \cup S2)]$. We then have $m'.X.C = (m.X.C - x.C) \cup (x'.C \cup y.C) = m.X.C \cup [(F1 \cup S1) \times \{p\}] \cup [\{p\} \times (F2 \cup S2)]$. Therefore, we have $m'.C = m'.cd.C \cup m'.X.C = m.cd.C \cup m.X.C \cup [(F1 \cup S1) \times \{p\}] \cup [\{p\} \times (F2 \cup S2)] = m.C \cup [(F1 \cup S1) \times \{p\}] \cup [\{p\} \times (F2 \cup S2)]$, i.e. $m'$ and $m$ have the same set of connections, except the connections with $p$. Since $p$ is not a primitive process, $m'$ and $m$ have the same set of connections with primitive processes.

Therefore, $m'$ and $m$ have the same intrinsic model, i.e. they are equivalent.

## 4 Conclusions and future work

In this paper, we have demonstrated the significance of restructuring levelled data-flow diagrams in structured analysis and design, and the necessity for providing specific restructuring operations by data-flow diagram editors. An eligible set of restructuring operations must meet all the needs of restructuring; preserve the properties of consistency and completeness; and guarantee that both system models of levelled data-flow diagrams are equivalent, before and after restructuring. We have proposed such a set and confirmed its eligibility.

As the concept of sets and relations is used to define the structures of data-flow diagrams and system models, it is easy to implement system models of levelled data-flow diagrams in relational databases [18, 19] and translate the formal specification of restructuring operators into procedures of data manipulation statements. Indeed, we have developed a data-flow diagram editor that provides all the proposed restructuring operators [20]. The editor puts special emphasis on the validation and preservation of the consistency and completeness of levelled data-flow diagrams. It provides users with a language to define their own formation rules; can check formation rules immediately; and can enforce formation rules, when requested, during editing operations. The editor also provides users with a language with which to define their own editing operators; the proposed restructuring operators are implemented in this definition language. The editor's first prototype is currently operational on an IBM PC/AT under the MS DOS operating system.

Two additions can be made to current work. One addition is to support the composition of data. Flows may carry primitive data, composite data or only part of the composite data. Split and merge of flows are then allowed. Stores may deposit primitive data, composite data, or only part of the composite data. A store of composite data can be accessed with full composition or with only part of the composition. Another improvement is to incorporate control flows, control processes and state transition diagrams. Such an addition is useful for modelling complex real-time systems [21, 22] and has been carried out by Ward and Mellor [23], Hatley and Pirbhai [15], and Yourdon [16] in their modelling tools. Investigation into restructuring operations for both additions is proposed for our future research.

## 5 References

[1] DEMARCO, T.: 'Structured analysis and system specification' (Prentice-Hall, 1979)

[2] GANE, C., and SARSON, T.: 'Structured systems analysis: tools and techniques' (Prentice-Hall, 1979)

[3] YOURDON, E., and CONSTANTINE, L.L.: 'Structured design: fundamentals of a discipline of computer program and system design' (Prentice-Hall, 1979)

[4] PAGE-JONES, M.: 'Practical guide to structured systems design' (Prentice-Hall, 1988), 2nd edn.

[5] BECK, L.L., and PERKINS, T.E.: 'A survey of software engineering practice: tools, methods, and results', *IEEE Trans.*, 1983, **SE-9,** (5), pp. 541–561

[6] NORMAN, R.J., and NUNAMAKER, J.F., Jr.: 'CASE productivity perceptions of software engineering professionals', *Commun. ACM*, 1989, **32,** (9), pp. 1102–1108

[7] DOCKER, T.W.G.: 'SAME — a structured analysis tool and its implementation in Prolog' *in* KOWALSKI, R.A., and BOWEN, K.A. (Eds.): 'Logic programming'. Proc. 5th Int. Conf. and Symp. on Logic Programming, Seattle, Washington, August 1988, pp. 82–95 (MIT Press, 1988)

[8] MODELL, H.S., and HERMENS, L.A.: 'More CASE tools on the Mac: Turbo CASE and MacBubbles', *IEEE Sofw.*, 1990, **7,** (1), pp. 133–139

[9] TSE, T.H., and PONG, L.: 'Towards a formal foundation for DeMarco data flow diagrams', *Comput. J.*, 1989, **32,** (1), pp. 1–12

[10] YANG, S.: 'Two CASE tools for the Macintosh', *IEEE Softw.*, 1989, **6,** (1), pp. 120–123

[11] TAN, K.P., CHUA, T.S., and LEE, P.T.: 'AUTO-DFD: an intelligent data flow processor', *Comput. J.*, 1989, **32,** (3), pp. 194–201

[12] RAMAMOORTHY, C.V., PRAKASH, A., TSAI, W.-T., and USUDA, Y.: 'Software engineering: problems and perspectives', *Computer*, 1984, **17,** (10), pp. 191–209

[13] WARD, P.T., and MELLOR, S.J.: 'Structured development for real-time systems: essential modeling techniques' (Yourdon Press, 1985) Vol. 2

[14] WARD, P.T., and MELLOR, S.J.: 'Structured development for real-time systems: implementation modeling techniques' (Yourdon Press, 1986) Vol. 3

[15] HATLEY, D.J., and PIRBHAI, I.A.: 'Strategies for real-time system specification' (Dorset House Publishing, 1987)

[16] YOURDON, E.: 'Modern structured analysis' (Prentice-Hall, 1989)

[17] MENDELSON, E.: 'Introduction to mathematical logic' (D. Van Nostrand Company, 1979) 2nd edn.

[18] CODD, E.F.: 'A relational model for large shared data banks', *Commun. ACM*, 1970, **13,** (6), pp. 377–387

[19] DATE, C.J.: 'An introduction to database systems. Vol I' (Addison-Wesley, 1986) 4th edn.

[20] CHEN, M.-J., and CHUNG, C.-G.: 'On the design of FLEDGED—a flexible editing tool for data flow diagrams'. Proc. 3rd Int. Conf. on Software Engineering and Knowledge Engineering, Skokie, Illinois, June 1991

[21] POST, J.: 'Application of a structured methodology to real-time industrial software development', *Softw. Eng. J.*, 1986, **1,** (6), pp. 222–235

[22] YOURDON, E.: 'Whatever happened to structured analysis?', *Datamation*, 1986, **32,** (11), pp. 133–138

[23] WARD, P.T., and MELLOR, S.J.: 'Structured development for real-time systems: introduction and tools' (Yourdon Press, 1985) Vol. 1

## Appendix 1

## Formal definitions of formation rules

A typical set of formation rules is formally defined below. Some variants are also described.

### *Appendix 1.1*

### *Formation rules concerning the hierarchial structure of system models*

A system model of levelled data-flow diagrams is a strictly hierarchical structure, with a context diagram defined at the top and all transformation diagrams defined in lower levels. At least one transformation diagram must be used to depict the data transformation details of the system (*Ha*). Every transformation diagram must depict the data transformation details of a process (*Hb*) but not two distinct processes at the same time (*Hc*). Nor must the same process be depicted by two distinct transformation diagrams (*Hd*). Furthermore, any transformation diagram must not depict any process in the diagram or in any other diagram decomposed directly or indirectly from any process in the diagram, i.e. any decomposition sequence is not cyclic (*He*). Listed below are all the formation rules concerning the hierarchical structure of a system model (*m*).

$Ha.\ m.X \neq \varnothing$
$Hb.\ (\forall x \in m.X)[(\exists p \in m.P)(\langle p, x\rangle \in m.D)]$
$Hc.\ (\forall x \in m.X)(\forall p, q \in m.P)[(\langle p, x\rangle, \langle q, x\rangle \in m.D)$

$$\Rightarrow (p = q)]$$

$$Hd.\ (\forall p \in m.P)(\forall x, y \in m.X)[(\langle p, x\rangle, \langle p, y\rangle \in m.D) \Rightarrow (x = y)]$$

$$He.\ \neg(\exists x_0, p_0, \ldots, x_n, p_n)[(1 \leqslant n) \wedge (x_0, \ldots, x_n \in m.X) \wedge (p_0 \in x_0.P) \wedge \cdots \wedge (p_n \in x_n.P) \wedge (\langle p_0, x_1\rangle, \ldots, \langle p_{n-1}, x_n\rangle, \langle p_n, x_0\rangle \in m.D)]$$

When a system model conforms to all of these formation rules, it is *hierarchically well formed.*

## *Appendix 1.2*

### *Formation rules concerning the structure of context diagrams*

The context diagram of a system model contains just one process representing all of the system's functions (*Ca*), all the terminators with which the system interacts, and any necessary flows between the process and the terminators; no stores are allowed (*Cb*). Only connections between the flows and the process, as well as between the flows and the terminators, are permitted (*Cc*). The process must have both flow inputs and outputs (*Cd*). Every terminator must be connected to at least one flow (*Ce*). A flow must not have the process both as its sender and receiver. (*Cf*). A flow must not have terminators, whether they are the same or not, both as its sender and receiver (*Cg*). Every flow must have both a sender and receiver (*Ch*), but at most one sender and one receiver (*Ci*). Listed below are all the formation rules concerning the structure of a context diagram (*cd*).

$$Ca.\ (\exists p)(cd.P = \{p\})$$
$$Cb.\ cd.S = \varnothing$$
$$Cc.\ cd.C \subseteq [cd.F \times (cd.P \cup cd.T)] \cup [(cd.P \cup cd.T) \times cd.F]$$
$$Cd.\ (\forall p \in cd.P)[(F^i(p) \neq \varnothing) \wedge (F^o(p) \neq \varnothing)]$$
$$Ce.\ (\forall t \in cd.T)[F^i(t) \cup F^o(t) \neq \varnothing]$$
$$Cf.\ (\forall p \in cd.P)[F^i(p) \cap F^o(p) = \varnothing]$$
$$Cg.\ F^i(cd.T) \cap F^o(cd.T) = \varnothing$$
$$Ch.\ [cd.F \subseteq F^i(cd.P \cup cd.T)] \wedge [cd.F \subseteq F^o(cd.P \cup cd.T)]$$
$$Ci.\ (\forall q, r \in cd.P \cup cd.T)[(q \neq r) \Rightarrow (F^i(q) \cap F^i(r) = \varnothing) \wedge (F^o(q) \cap F^o(r) = \varnothing)]$$

Note that rules *Cc* and *Ch* imply $cd.F = F^i(cd.P \cup cd.T) = F^o(cd.P \cup cd.T)$. This fact, together with rules *Ca*, *Cf* and *Cg*, implies that $F^i(cd.P) = F^o(cd.T)$, $F^o(cd.P) = F^i(cd.T)$, $F^i(cd.P) \cap F^i(cd.T) = \varnothing$, and $F^o(cd.P) \cap F^o(cd.T) = \varnothing$.

## *Appendix 1.3*

### *Formation rules concerning the structure of transformation diagrams*

A transformation diagram details the data transformation of a composite process by its component processes and their connected flows, accessed stores and connections. At least one process is required (*Ta*), but no terminator is allowed (*Tb*). Only connections between the flows and the processes, as well as between the stores and the processes, are permitted (*Tc*). Every process must have both inputs and outputs, whether they are flows or stores (*Td*), but at least one of them must be a flow (*Te*). A flow must not have the same process both as its sender and receiver (*Tf*). It is not necessary for a flow to have both its sender and receiver in the same transformation diagram; the open end will match that of the diagram's parent process. However, a flow must not pass through a transformation diagram without connecting with a process (*Tg*), or have more than one sender or receiver (*Th*). Stores need not have both their readers and writers in the same transformation diagram, but a store must not dangle in a transformation diagram without being accessed (*Ti*). Listed below are all the formation rules concerning the structure of a transformation diagram (*x*).

$$Ta.\ x.P \neq \varnothing$$
$$Tb.\ x.T = \varnothing$$
$$Tc.\ x.C \subseteq [(x.F \cup x.S) \times x.P] \cup [x.P \times (x.F \cup x.S)]$$
$$Td.\ (\forall p \in x.P)[(F^i(p) \cup S^i(p) \neq \varnothing) \wedge (F^o(p) \cup S^o(p) \neq \varnothing)]$$
$$Te.\ (\forall p \in x.P)[F^i(p) \cup F^o(p) \neq \varnothing]$$
$$Tf.\ (\forall p \in x.P)[F^i(p) \cap F^o(p) = \varnothing]$$
$$Tg.\ x.F \subseteq F^i(x.P) \cup F^o(x.P)$$
$$Th.\ (\forall p, q \in x.P)[(p \neq q) \Rightarrow (F^i(p) \cap F^i(q) = \varnothing) \wedge (F^o(p) \cap F^o(q) = \varnothing)]$$
$$Ti.\ x.S \subseteq S^i(x.P) \cup S^o(x.P)$$

Note that rules *Tc* and *Tg* imply that $x.F = F^i(x.P) \cup F^o(x.P)$, and that rules *Tc* and *Ti* imply that $x.S = S^i(x.P) \cup S^o(x.P)$. In those variants of structured analysis that use the Petri-net-based data-flow model to interpret the dynamic behaviour of a system model, every process is required to have both flow inputs and outputs, and so rules *Td* and *Te* must be changed into $(\forall p \in x.P)[(F^i(p) \neq \varnothing) \wedge (F^o(p) \neq \varnothing)]$. We do not use such conventions.

## *Appendix 1.4*

### *Formation rules concerning balancing*

Every transformation diagram represents a more detailed, but identical, view of the data transformation of its parent process. Net flow inputs and outputs of a transformation diagram must be identical to the flow inputs and outputs of its parent process. The conventions are specified in the *flow balancing rule.*

$$Ba.\ (\forall p \in m.P)(\forall x \in m.X)[(\langle p, x\rangle \in m.D) \Rightarrow (F^i(p) = F^i(x.P) - F^o(x.P)) \wedge (F^o(p) = F^o(x.P) - F^i(x.P))]$$

Similarly, the occurrences of stores in the transformation diagrams of a system model must also conform to conventions, called *store balancing rules.* They require that the highest level diagram where a store appears is where it is used as an interface between two processes (*Bb*); that a store accessed by a composite process must be a store accessed by some of its component processes with the same type of access (*Bc*); and that references of a composite process to a store must show all references of its component processes to the store (*Bd*). Thus

$$Bb.\ (\forall p \in m.P)(\forall x \in m.X)(\forall s)[(\langle p, x\rangle \in m.D) \wedge (x.S - (S^i(p) \cup S^o(p)) \neq \varnothing) \wedge (s \in x.S - (S^i(p) \cup S^o(p))) \Rightarrow (\exists q, r \in x.P)[(q \neq r) \wedge (s \in S^i(q) \cap S^o(r))]]$$
$$Bc.\ (\forall p \in m.P)(\forall x \in m.X)[(\langle p, x\rangle \in m.D) \Rightarrow (S^i(p) \subseteq S^i(x.P)) \wedge (S^o(p) \subseteq S^o(x.P))]$$
$$Bd.\ (\forall p \in m.P)(\forall x \in m.X)[(\langle p, x\rangle \in m.D) \Rightarrow (S^i(p) \cap S^o(x.P) \subseteq S^o(p)) \wedge (S^o(p) \cap S^i(x.P) \subseteq S^i(p))]$$

In those variants of conventions that do not require a store to have both readers and writers, the term $s \in S^i(p) \cap S^o(q)$ in rule *Bb* must be replaced by the term $s \in (S^i(p) \cup S^o(p)) \cap (S^i(q) \cup S^o(q))$. We do not use such conventions.

When every parent-child pair, of composite process and transformation diagram, in a system model conforms to all the flow and store balancing rules, the model is *in balance.* In a hierarchically well formed system model in balance, inputs and outputs of every transformation diagram match those of its parent process exactly.

### *Appendix 1.5*

### *Formation rules concerning the originality of elements*

The scope of every element in a system model extends to all the data-flow diagrams in the model. Any two distinct elements of the same kind are not allowed to have the same label. Within a data-flow diagram, the originality conventions of elements are automatically enforced because we define a data-flow diagram structure as four 'sets' instead of four 'bags' (see Definition 1).

However, for any two distinct diagrams in a system model, cross-references are required to enforce the originality conventions. Cross-references of terminators are not necessary because all terminators must reside in the context diagram. Cross-references of processes are trivial. We can guarantee the originality of processes by preventing any two distinct diagrams from having the same processes (*Ua*). Cross-references of flows and stores need some manipulation. The equivalence of flows and of stores, in a transformation diagram, to those connected with its parent process has been specified in the balancing rules. Thus, for any two diagrams with one's parent process residing in another diagram, to guarantee the originality of flows and stores between both diagrams requires the flows and stores in both diagrams to be different, except for those equivalent to each other as prescribed in the balancing rules.

This concept can be expended to any two distinct diagrams. For every transformation diagram, we distinguish between flows (stores) that are equivalent to flows (stores) connected with its parent process and those that are not, and designate them *balancing flows* (*stores*) and *local flows* (*stores*), respectively. To guarantee the originality of flows between a transformation diagram and the context diagram, local flows in the transformation diagram need to be different to any flows in the context diagram (*Ub*). Note that a context diagram has no store. To guarantee the originality of flows and stores between two distinct transformation diagrams, both diagrams must not have the same local flows or local stores (*Uc* and *Ud*). Listed below are all the formation rules concerning the originality of elements between any two distinct diagrams in a system model ($m$), with the assumption that the model is hierarchically well formed.

$$\begin{aligned}
&Ua.\ (\forall x, y \in m.X \cup \{m.cd\})[(x \neq y) \Rightarrow (x.P \cap y.P = \varnothing)]\\
&Ub.\ (\forall x \in m.X)[(x.F - (F^i(x.pa) \cup F^o(x.pa))) \cap m.cd.F\\
&\qquad = \varnothing]\\
&Uc.\ (\forall x, y \in m.X)[(x \neq y)\\
&\quad \Rightarrow (x.F - (F^i(x.pa) \cup F^o(x.pa))) \cap (y.F - (F^i(y.pa)\\
&\qquad \cup F^o(y.pa))) = \varnothing]\\
&Ud.\ (\forall x, y \in m.X)[(x \neq y)\\
&\quad \Rightarrow (x.S - (S^i(x.pa) \cup S^o(x.pa))) \cap (y.S - (S^i(y.pa)\\
&\qquad \cup S^o(y.pa))) = \varnothing]
\end{aligned}$$

Some readers may wonder why we do not require cross-references of the consistent connections of balancing flows and stores; the main reason is that the balancing rules not only specify the equivalence of flows and of stores, but also their consistent connections. Thus, cross-references of such consistent connections will be redundant.

### *Appendix 1.6*

### *Formation rules concerning the external interfaces of process groups*

Conceptually, any subset of primitive processes in a system model can be grouped together, representing a subsystem of the system. Every subsystem must interact with another part of the system or the terminators outside the system. A subsystem must have inputs from the externals and outputs to the externals (*Ea*), but it must not connect with the externals only through stores (*Eb*). Listed below are all the formation rules concerning the external interfaces of any group of primitive processes in a system model ($m$). (Let $\boldsymbol{m}.\mathbb{P}$ be the set of all the primitive processes in the system model $m$, i.e. $m.\mathbb{P} = \{p \in m.P \mid \neg(\exists z \in m.X)(\langle p, z\rangle \in m.D)\}$.)

$$\begin{aligned}
&Ea.\ (\forall Q)[(Q \subseteq m.\mathbb{P}) \ \wedge\ (Q \neq \varnothing)\\
&\quad \Rightarrow [(F^i(Q) \cap (F^o(m.T) \cup F^o(m.\mathbb{P} - Q)))\\
&\qquad \cup (S^i(Q) \cap (S^i(m.\mathbb{P} - Q) \cup S^o(m.\mathbb{P} - Q))) \neq \varnothing]\\
&\qquad \wedge\ [(F^o(Q) \cap (F^i(m.T) \cup F^i(m.\mathbb{P} - Q)))\\
&\qquad \cup (S^o(Q) \cap (S^i(m.\mathbb{P} - Q) \cup S^o(m - Q))) \neq \varnothing]]\\
&Eb.\ (\forall Q)[(Q \subseteq m.\mathbb{P}) \ \wedge\ (Q \neq \varnothing)\\
&\quad \Rightarrow [(F^i(Q) \cap (F^o(m.T) \cup F^o(m.\mathbb{P} - Q)))\\
&\qquad\quad \cup (F^o(Q) \cap (F^i(m.T) \cup F^i(m.\mathbb{P} - Q))) \neq \varnothing]]
\end{aligned}$$

In a system model that conforms to both rules, any composite process, with its inputs and outputs summarising the external interfaces of all the primitive processes in the diagrams decomposed directly or indirectly from itself, will have non-empty inputs and outputs, with one of them being a flow. For those variants of structured analysis that use the Petri-net-based data-flow model, rules *Ea* and *Eb* must be changed into $(\forall Q)[(Q \subseteq m.\mathbb{P}) \wedge (Q \neq \varnothing) \Rightarrow (F^i(Q) \cap (F^o(m.T) \cup F^o(m.\mathbb{P} - Q)) \neq \varnothing) \wedge (F^o(Q) \cap (F^i(m.T) \cup F^i(m.\mathbb{P} - Q)) \neq \varnothing)]$.

## Appendix 2

## Validation of the fold operator

In this Appendix, we validate that the system model after the operation of the *fold* operator, $m'$, is still structurally consistent and complete, i.e. it still conforms to all of the formation rules.

### *Appendix 2.1*

### *Validation for the hierarchical structure*

All the data-flow diagrams in a system model must form a strictly hierarchical structure as prescribed in rules *Ha–He*. Below, we show rule by rule that $m'$ conforms to all of the rules.

- Because $m'.X = m.X \cup \{y\} \neq \varnothing$, $m'$ abides by rule *Ha*.
- The only change to $m.X$ is the addition of the diagram $y$. As $m'.D = m.D \cup \{\langle p, y\rangle\}$, $y$ has a parent process $p$.

Therefore, $m'$ abides by rule *Hb*.

- The only change to $m.D$ is the addition of the entry $\langle p, y\rangle$. As $y$ is not in $m.X$, no $\langle \_, y\rangle$ is in $m.D$. Therefore, $m'$ abides by rule *Hc*.
- Similarly, as $p$ is not in $m.P$, no $\langle p, \_\rangle$ is in $m.D$. Therefore, $m'$ also abides by Rule *Hd*.
- As the only change to $m.D$ is the addition of the entry $\langle p, y\rangle$, the only changed decomposition sequences are: $\ldots$, $\langle x.pa, x\rangle$, $\langle r, \_\rangle$, $\ldots$, for every composite process $r$ in $R$. They are changed to $\ldots$, $\langle x.pa, x\rangle$, $\langle p, y\rangle$, $\langle r, \_\rangle$, $\ldots$ Such change does not make them cyclic. Therefore, $m'$ abides by rule *He*.

## *Appendix 2.2*

### *Validation for the structure of the context diagram*

Rules *Ca–Ci* are concerned with the formation of a context diagram. Since the *fold* operator does not change the context diagram, the context diagram in $m'$ conforms to rules *Ca–Ci* just as that in $m$ does.

## *Appendix 2.3*

### *Validation for the structure of transformation diagrams*

Except for $x'$ and $y$, all the transformation diagrams in $m'$ are the same as those in $m$. Since only all the processes in $R$, as well as only all their connections, are moved to the diagram $y$, $y$ should conform to rules *Ta–Ti*. Otherwise, the diagram $x$ will not conform to the rules. Below, we validate only the diagram $x'$ for rules *Ta–Ti*.

□ Because $x'.P = x.P \cup \{p\} - R \neq \varnothing$, $x'$ abides by rule *Ta*.

□ Because $x'.T = x.T = \varnothing$, $x'$ abides by rule *Tb*.

□ By the specification of $x'.C$, we have $F^i(x'.P) = F^i(x.P - R) \cup F^i(p)$

$$= F^i(x.P - R) \cup (F^i(R) - F^o(R))$$
$$= F^i(x.P) - (F^i(R) \cap F^o(R)).$$

As $x$ abides by rule *Tc*, we have $F^i(x.P) \subseteq x.F$. Therefore, $F^i(x'.P) \subseteq x.F - (F^i(R) \cap F^o(R))$. According to the specification of $x'.F$, we have $x'.F = x.F - (F^i(R) \cap F^o(R))$. Therefore, $F^i(x'.P) \subseteq x'.F$. Similarly, we have $F^o(x'.P) \subseteq x'.F$. According to the specification of $x'.C$, we have $S^i(x'.P) = S^i(x.P - R) \cup S^i(p)$

$$= S^i(x.P - R) \cup (S^i(R) \cap [S^i(x.P - R) \cup S^o(x.P - R) \cup S^i(x.pa)])$$
$$\subseteq [S^i(x.P - R) \cup S^o(x.P - R) \cup S^i(x.pa)].$$

As $x$ abides by rule *Tc*, we have $S^i(x.P) \subseteq x.S$ and $S^o(x.P) \subseteq x.S$. As $\langle x.pa, x\rangle$ abides by rule *Bc*, we have $S^i(x.pa) \subseteq S^i(x.P) \subseteq x.S$. Therefore, $[S^i(x.P - R) \cup S^o(x.P - R) \cup S^i(x.pa)] \subseteq x.S$. According to the specfication of $x'.S$, we have $x'.S = x.S - ([S^i(R) \cup S^o(R)] - [S^i(x.P - R) \cup S^o(x.P - R) \cup S^i(x.pa) \cup S^o(x.pa)])$. Because $[S^i(x.P - R) \cup S^o(x.P - R) \cup S^i(x.pa)] \cap ([S^i(R) \cup S^o(R)] - [S^i(x.P - R) \cup S^o(x.P - R) \cup S^i(x.pa) \cup S^o(x.pa)]) = \varnothing$, we have $[S^i(x.P - R) \cup S^o(x.P - R) \cup S^i(x.pa)] \subseteq x'.S$, i.e. $S^i(x'.P) \subseteq x'.S$. Similarly, we have $S^o(x'.P) \subseteq x'.S$. Therefore, $x'$ abides by rule *Tc*.

□ Because $x'.P = x.P \cup \{p\} - R$ and the *fold* operator does not change the connections with any process in $x.P - R$, only process $p$ needs validation for rules *Td–Tf*. As $p$ is the parent process of the diagram $y$, to show that $p$ has both inputs and outputs, we first show that the interfaces of $p$ summarise all the external interfaces of the processes in $y.P$, which equals $R$. According to the specification of $x'.C$, we have $F^i(p) = F^i(R) - F^o(R)$ and $F^o(p) = F^o(R) - F^i(R)$. Therefore, the flow inputs and outputs of $p$ summarise all the external flow inputs and outputs of $y.P$. According to the specification of $x'.C$, we also have $S^i(p)$

$$= S^i(R) \cap [S^i(x.P - R) \cup S^o(x.P - R) \cup S^i(x.pa)]$$
$$= [S^i(R) \cap (S^i(x.P - R) \cup S^o(x.P - R))] \cup [S^i(R) \cap S^i(x.pa)]$$

The first term summarises all the stores both accessed by $x.P - R$ and read by $R$. As $S^i(x.pa)$ summarises all the external store inputs of $x.P$ and $x.P = (x.P - R) \cup R$, the second term, $S^i(R) \cap S^i(x.pa)$, summarises all the stores both accessed by the externals of $x.P$ and read by $R$. Therefore, $S^i(p)$ summarises all the external store inputs of $y.P$. Similarly, $S^o(p)$ summarises all the external store outputs of $y.P$. Because the interfaces of $p$ summarise all the external interfaces of the processes in $y.P$, they summarise all the external interfaces of both the primitive processes in $y.P$ and those decomposed directly or indirectly from the composite processes in $y.P$. As $m$ abides by rule *Ea*, the external inputs or outputs of any non-empty subset of primitive processes in $m$ cannot be empty. Therefore, $x'$ abides by rule *Td*.

□ As $m$ abides by rule *Eb*, one of the external interfaces of any non-empty subset of primitive processes in $m$ must be a flow. Therefore, $x'$ abides by rule *Te*.

□ $x'.P = x.P \cup \{p\} - R$. As $x$ abides by rule *Tf*, $F^i(q) \cap F^o(q) = \varnothing$ for every $q$ in $x.P - R$. Furthermore, $F^i(p) \cap F^o(p) = (F^i(R) - F^o(R)) \cap (F^o(R) - F^i(R)) = \varnothing$. Therefore, $x'$ abides by rule *Tf*.

□ Uniting $F^i(x'.P)$ and $F^o(x'.P)$, we have $F^i(x'.P) \cup F^o(x'.P)$

$$= [F^i(x.P - R) \cup (F^i(R) - F^o(R))] \cup [F^o(x.P - R) \cup (F^o(R) - F^i(R))]$$
$$= (F^i(x.P) \cup F^o(x.P)) - (F^i(R) \cap F^o(R))$$

According to the specification of $x'.F$, we have $x'.F = x.F - (F^i(R) \cap F^o(R))$. As $x$ abides by rule *Tg*, we have $x.F \subseteq F^i(x.P) \cup F^o(x.P)$. Subtracting $F^i(R) \cap F^o(R)$ from both sides, we have $x.F - (F^i(R) \cap F^o(R)) \subseteq (F^i(x.P) \cup F^o(x.P)) - (F^i(R) \cap F^o(R))$, i.e. $x'.F \subseteq F^i(x'.P) \cup F^o(x'.P)$. Therefore, $x'$ abides by rule *Tg*.

□ We divide $x'.P = x.P \cup \{p\} - R$ into two sets $x.P - R$ and $\{p\}$. As $x$ abides by rule *Th*, $F^i(q) \cap F^i(r) = \varnothing$ for any two distinct processes $q$, $r$ in $x.P - R$ or $q$ in $x.P - R$ and $r$ in $R$. For every $q$ in $x.P - R$, we have $F^i(p) \cap F^i(q) = (F^i(R) - F^o(R)) \cap F^i(q)$. The intersection must be empty; otherwise, a process $r$ in $R$ will exist such that $F^i(r) \cap F^i(q) \neq \varnothing$. Therefore, for any two distinct $q$, $r$ in $x'.P$, $F^i(q) \cap F^i(r) = \varnothing$. Similarly, for any two distinct $q$, $r$ in $x'.P$, $F^o(q) \cap F^o(r) = \varnothing$. Therefore, $x'$ abides by rule *Th*.

□ Uniting $S^i(x'.P)$ and $S^o(x'.P)$, we have $S^i(x'.P) \cup S^o(x'.P)$

$$= [S^i(x.P - R) \cup (S^i(R) \cap [S^i(x.P - R) \cup S^o(x.P - R) \cup S^i(x.pa)])]$$
$$\cup [S^o(x.P - R) \cup (S^o(R) \cap [S^i(x.P - R) \cup S^o(x.P - R)$$

$$\begin{aligned} &\qquad\qquad \cup S^o(x.pa)]) ] \\ &= [S^i(x.P-R) \cup S^o(x.P-R)] \cup [S^i(R) \cap S^i(x.pa)] \\ &\qquad \cup [S^o(R) \cap S^o(x.pa)]. \end{aligned}$$

According to the specification of $x'.S$, we have $x'.S = x.S - ([S^i(R) \cup S^o(R)] - [S^i(x.P-R) \cup S^o(x.P-R) \cup S^i(x.pa) \cup S^o(x.pa)])$. As $x$ abides by rule *Ti*, we have $x.S \subseteq S^i(x.P) \cup S^o(x.P)$. Therefore, we have $x'.S$

$$\begin{aligned} &\subseteq [S^i(x.P) \cup S^o(x.P)] - ([S^i(R) \cup S^o(R)] \\ &\quad - [S^i(x.P-R) \cup S^o(x.P-R) \\ &\quad \cup S^i(x.pa) \cup S^o(x.pa)]) \\ &= ([S^i(x.P-R) \cup S^o(x.P-R) \cup S^i(R) \cup S^o(R)] \\ &\qquad - [S^i(R) \cup S^o(R)]) \\ &\quad \cup ([S^i(R) \cup S^o(R)] \cap [S^i(x.P-R) \cup S^o(x.P-R)]) \\ &\quad \cup ([S^i(R) \cup S^o(R)] \cap [S^i(x.pa) \cup S^o(x.pa)]) \\ &= [S^i(x.P-R) \cup S^o(x.P-R)] \cup [S^i(R) \cap S^i(x.pa)] \\ &\qquad \cup [S^o(R) \cap S^o(x.pa)] \\ &\quad \cup [S^i(R) \cap S^o(x.pa)] \cup [S^o(R) \cap S^i(x.pa)]. \end{aligned}$$

As $\langle x.pa, x \rangle$ abides by rule *Bd*, we have $S^i(x.P) \cap S^o(x.pa) \subseteq S^i(x.pa)$. Intersecting both sides by $S^i(R)$, we have $S^i(R) \cap S^i(x.P) \cap S^o(x.pa) \subseteq S^i(R) \cap S^i(x.pa)$. Because $S^i(R) \subseteq S^i(x.P)$, we have $S^i(R) \cap S^o(x.pa) \subseteq S^i(R) \cap S^i(x.pa)$. Similarly, we have $S^o(R) \cup S^i(x.pa) \subseteq S^o(R) \cap S^o(x.pa)$. Subsequently, $x'.S \subseteq [S^i(x.P-R) \cup S^o(x.P-R)] \cup [S^i(R) \cap S^i(x.pa)] \cup [S^o(R) \cap S^o(x.pa)]$, i.e. $x'.S \subseteq S^i(x'.P) \cup S^o(x'.P)$. Therefore, $x'$ abides by rule *Ti*.

## *Appendix 2.4*

### *Validation for balancing*

Balancing rules require that every parent-child pair of composite process and transformation diagram are balanced with each other by common connected flows and stores. The *fold* operator moves the set of processes $R$ and all their connections from the diagram $x$ into the newly created diagram $y$, but leaves alone the connections of any process in $x.P - R$. Therefore, we only need to validate the balancing of the two pairs $\langle p, y \rangle$ and $\langle x'.pa, x' \rangle$. We first validate the balancing of $\langle p, y \rangle$ as follows.

- According to the specfication of $y.C$, we have $F^i(y.P) = \bigcup_{r \in R} F^i(r) = F^i(R)$ and $F^o(y.P) = \bigcup_{r \in R} F^o(r) = F^o(R)$. According to the specification of $x'.C$, we have $F^i(p) = F^i(R) - F^o(R)$ and $F^o(p) = F^o(R) - F^i(R)$. Therefore, $F^i(p) = F^i(y.P) - F^o(y.P)$ and $F^o(p) = F^o(y.P) - F^i(y.P)$, i.e. $\langle p, y \rangle$ abides by the flow balancing rule, rule *Ba*.
- According to the specification of $y.S$ and $x.C$, the set of local stores in $y$ is

$$\begin{aligned} &y.S - (S^i(p) \cup S^o(p)) \\ &\quad = [S^i(R) \cup S^o(R)] - [(S^i(R) \cap [S^i(x.P-R) \\ &\qquad\qquad \cup S^o(x.P-R) \cup S^i(x.pa)]) \\ &\qquad \cup (S^o(R) \cap [S^i(x.P-R) \cup S^o(x.P-R) \\ &\qquad \cup S^o(x.pa)])] \\ &\quad = [S^i(R) \cup S^o(R)] - [(S^i(R) \cup S^o(R)) \\ &\qquad\qquad \cap (S^i(x.P-R) \cup S^o(x.P-R))] \\ &\qquad - [(S^i(R) \cup S^o(R)) \cap (S^i(x.pa) \cup S^o(x.pa))] \\ &\quad = [S^i(R) \cup S^o(R)] - [(S^i(R) \cup S^o(R)) \\ &\qquad\qquad \cap (S^i(x.P-R) \cup S^o(x.P-R))] \\ &\qquad - (S^i(x.pa) \cup S^o(x.pa)) \\ &\quad \subseteq x.S - (S^i(x.pa) \cup S^o(x.pa)) \end{aligned}$$

i.e. every local store in $y$ is a local store in $x$. Because the term $(S^i(R) \cup S^o(R)) \cap (S^i(x.P-R) \cup S^o(x.P-R))$ has been subtracted, local stores in $y$ are local stores in $x$ without connection with any process in $x.P - R$. As $x$ abides by rule *Bb*, every local store in $x$ will have a reader and a writer, not the same one, in $x$. Therefore, both reader and writer of every local store in $y$ must be in $R$, i.e. $y.P$, namely, $\langle p, y \rangle$ abides by rule *Bb*.
- According to the specification of $x'.C$, we have $S^i(p) = S^i(R) \cap [S^i(x.P-R) \cup S^o(x.P-R) \cup S^i(x.pa)] \subseteq S^i(R)$. According to the specification of $y.C$, we have $S^i(y.P) = \bigcup_{r \in R} S^i(r) = S^i(R)$. Therefore, $S^i(p) \subseteq S^i(y.P)$. Similarly, we have $S^o(p) \subseteq S^o(y.P)$. Therefore $\langle p, y \rangle$ abides by rule *Bc*.
- According to the specification of $y.C$, we have $S^o(y.P) = \bigcup_{r \in R} S^o(r) = S^o(R)$. Intersecting $S^i(p)$ and $S^o(y.P)$, we have

$$\begin{aligned} &S^i(p) \cap S^o(y.P) \\ &\quad = (S^i(R) \cap [S^i(x.P-R) \cup S^o(x.P-R) \cup S^i(x.pa)]) \\ &\qquad\qquad \cap S^o(R) \\ &\quad \subseteq S^o(R) \cap [S^i(x.P-R) \cup S^o(x.P-R) \cup S^i(x.pa)] \\ &\quad = S^o(R) \cap [S^i(x.P-R) \cup S^o(x.P-R) \cup (S^i(x.pa) \\ &\qquad\qquad \cap S^o(R))] \\ &\quad \subseteq S^o(R) \cap [S^i(x.P-R) \cup S^o(x.P-R) \cup (S^i(x.pa) \\ &\qquad\qquad \cap S^o(x.P))]. \end{aligned}$$

As $\langle x.pa, x \rangle$ abides by rule *Bd*, we have $S^i(x.pa) \cap S^o(x.P) \subseteq S^o(x.pa)$. Thus, $S^i(p) \cap S^o(y.P) \subseteq S^o(R) \cap [S^i(x.P-R) \cup S^o(x.P-R) \cup S^o(x.pa)] = S^o(p)$, specified in $x.C$. Similarly, we have $S^o(p) \cap S^i(y.P) \subseteq S^i(p)$. Therefore, $\langle p, y \rangle$ abides by rule *Bd*.

Thus, we have validated the balancing of the pair $\langle p, y \rangle$. Now, we validate the balancing of $\langle x'.pa, x' \rangle$ in the following.

□ According to the specification of $x'C$, we have $F^i(x'.P) = (F^i(x.P) - F^i(R)) \cup F^i(p) = (F^i(x.P) - F^i(R)) \cup (F^i(R) - F^o(R)) = F^i(x.P) - (F^i(R) \cap F^o(R))$. Similarly, we have $F^o(x'.P) = F^o(x.P) - (F^i(R) \cap F^o(R))$. Subtracting the two equations from each other, we obtain $F^i(x'.P) - F^o(x'.P) = F^i(x.P) - F^o(x.P)$ and $F^o(x'.P) - F^i(x'.P) = F^o(x.P) - F^i(x.P)$. As $\langle x.pa, x \rangle$ abides by rule *Ba*, we have $F^i(x.pa) = F^i(x.P) - F^o(x.P)$ and $F^o(x.pa) = F^o(x.P) - F^i(x.P)$. Thus, $F^i(x'.P) - F^o(x'.P) = F^i(x.pa)$ and $F^o(x'.P) - F^i(x'.P) = F^o(x.pa)$. Because the *fold* operator does not change any connection with $x.pa$, we have $F^i(x'.pa) = F^i(x.pa)$ and $F^o(x'.pa) = F^o(x.pa)$. Therefore, $F^i(x'.P) - F^o(x'.P) = F^i(x'.pa)$ and $F^o(x'.P) - F^i(x'.P) = F^o(x'.pa)$, i.e. $\langle x'.pa, x' \rangle$ abides by the flow balancing rule *Ba*.

□ Rule *Bb* requires that every local store of a transformation diagram is accessed by at least two processes in the diagram, one as a reader and the other as a writer. Consider a local store, say $s$, of the diagram $x$, which has been changed to $x'$ and $y$. There are three cases. The first is that the two processes accessing $s$ are both in $x.P - R$. This is no problem because every process in $x.P - R$ is in $x'.P$. The second case is that one process is in $x.P - R$ and the other process is in $R$. To abide by rule *Bb*, the store $s$ must be moved all the processes in $R$ to $y$, and $p$, residing in $x'$, is the parent process of $y$). This requirement is true because $S^i(R) \cap [S^i(x.P-R) \cup S^o(x.P-R)] \subseteq S^i(R) \cap [S^i(x.P-R) \cup S^o(x.P-R) \cup S^i(x.pa)] = S^i(p)$ and $S^o(R) \cap [S^i(x.P -$

$R) \cup S^o(x.P - R)] \subseteq S^o(R) \cap [S^i(x.P - R) \cup S^o(x.P - R) \cup S^o(x.pa)] = S^o(p)$. The third case is that all the processes accessing $s$ are in $R$. If the store $s$ is not connected with the parent process of $x$, the store must be a local store of the diagram $y$, i.e. it must not be connected with $p$, the parent process of $y$. This requirement is true because $[S^i(R) \cup S^o(R)] - [S^i(x.P - R) \cup S^o(x.P - R)] - [S^i(x.pa) \cup S^o(x.pa)] = y.S - (S^i(p) \cup S^o(p))$, the right side of which is the set of local stores in $y$. (The reason why the left side is equal to the right side has been shown above.) By the three cases, we conclude that $\langle x'.pa, x' \rangle$ abides by rule *Bb*.

□ According to the specification of $x'.C$, we have $S^i(x'.P)$

$$\begin{aligned}
&= S^i(x.P - R) \cup (S^i(R) \cap [S^i(x.P - R) \cup S^o(x.P - R) \\
&\qquad \cup S^i(x.pa)]) \\
&= S^i(x.P) \cap [S^i(x.P - R) \cup S^o(x.P - R) \cup S^i(x.pa)] \\
&\supseteq S^i(x.P) \cap S^i(x.pa).
\end{aligned}$$

As $\langle x.pa, x \rangle$ abides by rule *Bc*, we have $S^i(x.pa) \subseteq S^i(x.P)$. Then, $S^i(x.P) \cap S^i(x.pa) = S^i(x.pa)$, the right side of which equals $S^i(x'.pa)$ as the *fold* operator does not change any connection with $x.pa$. Therefore, $S^i(x'.pa) \subseteq S^i(x'.P)$. Similarly, we have $S^o(x'.pa) \subseteq S^o(x'.P)$. Therefore, $\langle x'.pa, x' \rangle$ abides by rule *Bc*.

□ According to the specification of $x'.C$, we have $S^i(x'.pa) \cap S^o(x'.P)$

$$\begin{aligned}
&= S^i(x.pa) \cap [S^o(x.P - R) \cup (S^o(R) \cap [S^i(x.P - R) \\
&\qquad \cup S^o(x.P - R) \cup S^o(x.pa)])] \\
&= S^i(x.pa) \cap (S^o(x.P) \cap [S^i(x.P - R) \cup S^o(x.P - R) \\
&\qquad \cup S^o(x.pa)]) \\
&\subseteq S^i(x.pa) \cap S^o(x.P).
\end{aligned}$$

As $\langle x.pa, x \rangle$ abides by rule *Bd*, we have $S^i(x.pa) \cap S^o(x.P) \subseteq S^o(x.pa)$, the right side of which equals $S^o(x'.pa)$. Therefore, $S^i(x'.pa) \cap S^o(x'.P) \subseteq S^o(x'.pa)$. Similarly, we have $S^o(x'.pa) \cap S^i(x'.P) \subseteq S^i(x'.pa)$. Therefore, $\langle x'.pa, x' \rangle$ abides by rule *Bd*.

## *Appendix 2.5*

## *Validation for the originality of elements*

Since the *fold* operator has partitioned the diagram $x$ into $x'$ and $y$, in order to demonstrate that $m'$ conforms to the originality conventions of elements, we must consider cross-references between $x'$ and $y$, between $x'$ and $z$'s, and between $y$ and $z$'s, when $z$ is a diagram in $m'$ other than $x'$ and $y$.

- According to the specfication of $x'.P$ and $y.P$, we have $x'.P = x.P \cup \{p\} - R$ and $y.P = R$. Intersecting both, we have $x'.P \cap y.P = (x.P \cup \{p\} - R) \cap R = \varnothing$. Uniting both, we have $x'.P \cup y.P = (x.P \cup \{p\} - R) \cup R = x.P \cup \{p\}$. Note that $p \notin m.P$, and so $p \notin z.P$ for every $z \in m.X \cup \{m.cd\}$. As $x$ abides by rule *Ua*, we have $(x.P \cap z.P) = \varnothing$ for every $z \in m.X \cup \{m.cd\} - \{x\}$, *i.e.* $m'.X \cup \{m'.cd\} - \{x', y\}$. Therefore, $x'.P \cap z.P = \varnothing$ and $y.P \cap z.P = \varnothing$ for every $z \in m'.X \cup \{m'.cd\} - \{x', y\}$. As a result, $m'$ abides by rule *Ua*.
- According to the specification of $x'.F$, the set of local flows in $x'$ is $x'.F - (F^i(x'.pa) \cup F^o(x'.pa)) = x.F - (F^i(R) \cap F^o(R)) - (F^i(x.pa) \cup F^o(x.pa))$. According to the specification of $y.F$ and $x'.C$, the set of local flows in $y$ is $y.F - (F^i(p) \cup F^o(p)) = (F^i(R) \cup F^o(R)) - ((F^i(R) - F^o(R)) \cup (F^o(R) - F^i(R))) = F^i(R) \cap F^o(R)$. Because $(F^i(R) \cap F^o(R)) \cap (F^i(x.pa) \cup F^o(x.pa)) = \varnothing$, uniting the two sets of local flows, we obtain the set $x.F - (F^i(x.pa) \cup F^o(x.pa))$, which is the set of local flows in $x$. As $m$ abides by rule *Ub* and $x \in m.X$, local flows in $x$ are different to any flows in the context diagram. Subsequently, local flows in $x'$ and in $y$ are different to any flows in the context diagram. Therefore, $m'$ abides by rule *Ub*.
- We are aware that the union of the two sets of local flows in $x'$ and in $y$ equals the set of local flows in $x$. Since $m$ abides by rule *Uc* and $x \in m.X$, local flows in $x$ are different to local flows in any transformation diagram in $m$, other than $x$. In addition, $m'.X = m.X \cup \{x', y\} - \{x\}$. Therefore, local flows in $x'$ and in $y$ are different to local flows in any transformation diagram in $m'$, other than $x'$ and $y$. Furthermore, intersecting the two sets of local flows in $x'$ and in $y$, we have $[x.F - (F^i(R) \cap F^o(R)) - (F^i(x.pa) \cup F^o(x.pa))] \cap [F^i(R) \cap F^o(R)] = \varnothing$, i.e. $x'$ and $y$ have no common local flows. Therefore, $m'$ abides by rule *Uc*.
- According to the specification of $x'.S$, the set of local stores in $x'$ is

$$\begin{aligned}
&x'.S - (S^i(x'.pa) \cup S^o(x'.pa)) \\
&\quad = x.S - ([S^i(R) \cup S^o(R)] - [S^i(x.P - R) \\
&\qquad\qquad \cup S^o(x.P - R) \cup S^i(x.pa) \cup S^o(x.pa)]) \\
&\qquad - (S^i(x.pa) \cup S^o(x.pa)) \\
&\quad = (x.S - [S^i(R) \cup S^o(R)]) \cup ([S^i(R) \cup S^o(R)] \\
&\qquad\qquad \cap [S^i(x.P - R) \cup S^o(x.P - R)]) \\
&\qquad \cup ([S^i(R) \cup S^o(R)] \cap [S^i(x.pa) \\
&\qquad\qquad \cup S^o(x.pa)]) - (S^i(x.pa) \cup S^o(x.pa)) \\
&\quad = (x.S - [S^i(R) \cup S^o(R)]) \cup ([S^i(R) \cup S^o(R)] \\
&\qquad\qquad \cap [S^i(x.P - R) \cup S^o(x.P - R)]) \\
&\qquad - (S^i(x.pa) \cup S^o(x.pa)) \\
&\quad \subseteq x.S - (S^i(x.pa) \cup S^o(x.pa))
\end{aligned}$$

i.e. every local store in $x'$ is a local store in $x$. We are aware that every local store in $y$ is a local store in $x$. As $m$ abides by rule *Ud* and $x \in m.X$, local stores in $x$ are different to local stores in any transformation diagram in $m$, other than $x$. In addition, $m'.X = m.X \cup \{x', y\} - \{x\}$. Therefore, local stores in $x'$ and in $y$ are different to any local stores in any transformation diagram in $m'$, other than $x'$ and $y$. Furthermore, intersecting the two sets of local stores in $x'$ and in $y$, we have $[x'S - (S^i(x'.pa) \cup S^o(x'.pa))] \cap [y.S - (S^i(p) \cup S^o(p))] = [(x.S - [S^i(R) \cup S^o(R)]) \cup ([S^i(R) \cup S^o(R)] \cap [S^i(x.P - R) \cup S^o(x.P - R)]) - (S^i(x.pa) \cup S^o(x.pa))] \cap [[S^i(R) \cup S^o(R)] - [(S^i(R) \cup S^o(R)) \cap (S^i(x.P - R) \cup S^o(x.P - R))] - (S^i(x.pa) \cup S^o(x.pa))] = \varnothing$, i.e. $x'$ and $y$ have no common local stores. Therefore, $m'$ abides by rule *Ud*.

## *Appendix 2.6*
## *Validation for the external interfaces of process groups*

Rules *Ea* and *Eb* are concerned with the external interfaces of any subset of *primitive* processes in a system model. Since the *fold* operator neither deletes, creates any primitive processes, nor changes any connections with primitive processes in $m$, $m'$ conforms to rules *Ea* and *Eb*, as does $m$.

The paper was first received on 13th August 1990 and in revised form on 4th February 1991.

The authors are with the Institute of Computer Science and Information Engineering, National Chiao Tung University, Hsinchu, Taiwan 30050, Republic of China.