19/3/98

# Chapter 8

## Refinement testing.

*Summary. The theory of refinement testing. The refinement testing method.*

In the previous chapter we gave the theoretical basis of a functional testing method for systems specified as X-machines. The method generates test sets from the specification that ensure that the ensure that the system is fault-free provided that it is made of fault-free components and meets some “design for testing” requirements.

In practice, however, specifications of complex systems are seldom constructed from scratch. Instead, a process of refinement is used. This involves building several intermediary models $M_1, M_2, ..., M_n$, where $M_{i+1}$ is a refinement of some sort of $M_i$ and $M_n$ is the final system specification. Obviously, once the complete specification $M_n$ exists, we can always test its implementation using the Stream X-Machine Testing (SXMT) method. However, for large scale systems this approach is not desirable since a single X-machine specification is difficult to handle. Also, the extremely large state space of such a machine will result in an unmanageable test set. Instead, a more convenient approach would be to construct the test set incrementally by generating test sets for all intermediary models. In doing this, one would hope that the test set of the *i*'th model can be reused in the generation of that of the (*i*+1)'th model, so that the construction of the complete test set will also be a refinement process. This chapter will address this problem in the context of the state refinement defined in chapter 6.

### 8.1. Theoretical basis of refinement testing.

First, let us describe clearly the problem and the approach employed.
Let $M_1 = (Input, Output, Q_1, Memory, \Phi, \mathbf{F}_1, q_{01}, m_0)$ and
$M_1' = (Input', Output', Q_1', Memory', \Phi', \mathbf{F}_1', q_{01}', m_0')$ be two stream X-machine specifications of a system so that $M_1'$ is a state refinement of $M_1$ w.r.t. $(u, v)$. Then our aim is to generate an input set that tests $M_1'$ against its implementation.

We assume that this implementation can be modelled as a stream X-machine
$M_2' = (Input', Output', Q_2', Memory', \Phi', \mathbf{F}_2', q_{02}', m_0')$ with the same type $\Phi'$ as $M_1'$. Furthermore, we will assume that $M_2'$ can be expressed as a state refinement w.r.t. $(u, v)$ of a stream X-machine $M_2$ and that $M_2$ has the same type, $\Phi$, as $M_1$.

The way in which these requirements can be enforced on the system implementation will be discussed later. The arrangement is illustrated in Figure 8.1.

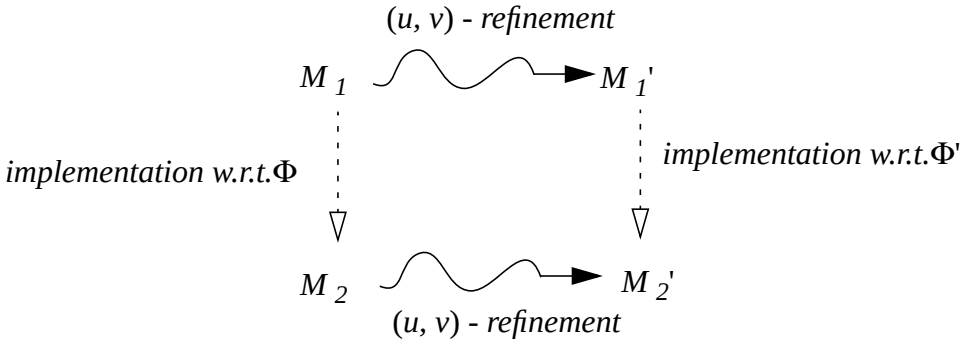



**Figure 8.1 A refinement and implementation relationship.**

Our approach will be to implement and test separately the refining machines used in the construction of $M_1'$ and to test the whole system for integration using a test set that is an augmentation of the test set of $M_1$ and $M_2$.

Before we go any further let us introduce the following concept.

**Definition** 8.1.1.
Let $A$ and $A'$ be two finite alphabets so that seq($A'$) covers seq($A$) w.r.t. $u$ and let $Y \subseteq$ seq($A$), $Y' \subseteq$ seq($A'$) be two sets. Then we say that $Y'$ *refines* $Y$ *w.r.t.* $u$ if the following is true.

$$\forall\, a^* \in Y\,, \exists\, a^{*\prime} \in Y' \text{ with } u^*(a^{*\prime}) = a^*$$

Recall that $u^*$ ,(Definition 6.3.1.1), takes several applications of $u$ and concatenates them together in $A$, so that $a^{*\prime}$ is a concatenation of a finite sequence of elements from $Y'$ .

If we denote by $u^{-1}: A \rightarrow$ seq($A'$) a function for which $u \circ u^{-1}$ is the identity and by $u^{-1*}$: seq($A$) $\rightarrow$ seq($A'$) the sequential generalisation of $u^{-1}$, then for any $Y \subseteq$ seq($A$), $Y' = u^{-1*}(Y)$ refines $Y$.

**Example** 8.1.1.
If for $n \geq 0$, $u_n$: seq(CHARS $\cup$ {*backspace*, *enter*}) $\rightarrow$ $\text{WORDS}_n$ is as in example 6.3.1.2 then $Y'$ refines $Y$ w.r.t. $u_2$, and $Z'$ refines $Y$ w.r.t. $u_2$ where:
$Y$ = {< >, ‘1’, ‘1::1’, ‘1::0’ ‘1::1’, ‘1::0’ ‘1::0’ ‘1::1’} and
$Y'$ = {< >, 1 :: *enter*, 1 ::1 :: *enter*, 1 :: 0 :: *enter* ::1::1 :: *enter*,
1 :: 0 :: *enter* :: 1 :: 0 :: *enter* ::1 :: 1 :: *enter*}
$Z'$ = {< >, 1 :: 0 :: *backspace* :: *enter*, 1 :: 1 :: 0 :: *backspace* :: *enter*,
1 :: 0 :: 0 :: *backspace* :: *enter* :: 1 :: 1 :: 0 :: *backspace* :: *enter*,
1 :: 0 :: 0 :: *backspace* :: *enter* :: 1 :: 0 :: 0 :: *backspace* :: *enter* :: 1 :: 1 :: 0 :: *backspace* :: *enter*}

Let us now introduce the notation we shall be using in this section and state the assumptions we shall be making.

Let $M_1 = (Input, Output, Q_1, Memory, \Phi, \mathbf{F}_1, q_{01}, m_0)$ and
$M_2 = (Input, Output, Q_2, Memory, \Phi, \mathbf{F}_2, q_{02}, m_0)$ be two stream X-machines with the same type, $\Phi$ and initial memory, $m_0$. Let $M_1' = (Input', Output', Q_1', Memory', \Phi', \mathbf{F}_1', q_{01}', m_0')$ and $M_2' = (Input', Output', Q_2', Memory', \Phi', \mathbf{F}_2', q_{02}', m_0')$ be state refinements of $M_1$ and $M_2$ respectively w.r.t. $(u, v)$ that also have the same type, $\Phi'$ and initial memory, $m_0'$. For simplicity we will consider that the key states of $M_1'$ and $M_2'$ respectively are $K_1 = Q_1$ and $K_2 = Q_2$, hence $q_{01}' = q_{01}$ and $q_{02}' = q_{02}$. This will not affect the generality of the results proven. We denote by $A_1, A_2, A_1'$ and $A_2'$ the associated automata of $M_1, M_2, M_1'$ and $M_2'$ respectively and $f_1, f_2, f_1', f_2'$ the functions computed by these machines.

Let $\{M_1'(q_1) : q_1 \in Q_1\}$ be the refining set of $M_1'$, with for any $q_1 \in Q_1$ ,
$M_1'(q_1) = (Input', Output', R_{q1}, Memory', \Phi', \mathbf{F}_{q1}, q_1, m_{q1}', T_{q1})$. For any $q_1 \in Q_1$ we denote by $A_1'(q_1)$ the associated automaton of $M_1'(q_1)$. Similarly $\{M_2'(q_2) : q_2 \in Q_2\}$ is the refining set with for any $q_2 \in Q_2$ , $M_2'(q_2) = (Input', Output', R_{q2}, Memory', \Phi', \mathbf{F}_{q2}, q_2, m_{q2}', T_{q2})$ and $A_2'(q_2)$ the associated automaton of $M_2'(q_2)$.

Note that the type of all refining machines was considered to be $\Phi'$. This is possible since not all processing functions have to be used as arc labels.

We shall assume that the following requirements are met.
1. $A_1$ is a minimal finite state machine.
2. The types $\Phi$ and $\Phi'$ are t-complete w.r.t. *Memory* and *Memory*' respectively and output-distinguishable.
3. For any $q_1 \in Q_1$ , $A_1'(q_1)$ is an accessible finite state machine.
4. All refinement machines have been tested against their corresponding implementations using the SXMT method. Thus for any $q_2 \in Q_2$ there exists $q_1 \in Q_1$ so that $A_1'(q_1)$ and $A_2'(q_2)$ are equivalent.

Also, for simplicity we shall assume that $A_2$ is accessible and for any $q_2 \in Q_2$ ,$A_2'(q_2)$ is accessible. This will not reduce in any way the generality of our results since the accessible part of $A_2'$ - and thus the input/output behaviour of $M_2'$ - will not be affected by this assumption.

**Lemma** 8.1.1.
Let $M_1, M_2, M_1', M_2'$ be four stream X-machines as described above. Then $A_1'$ and $A_2'$ are equivalent if there exists $e$: $A_2 \rightarrow A_1$ a surjective finite state machine morphism such that for any $q_2 \in Q_2$,$A_2'(q_2)$ and $A_1'(e(q_2))$ are equivalent.

*Proof*:
We define a relation $e'$: $Q_2' \leftrightarrow Q_1'$ by
$q_2'\ e'\ q_1'$ if and only if there exist $q_2 \in Q_2$, $q_1 \in Q_1$ with $e(q_2) = q_1$ and $q_2'$ and $q_1'$ are right-congruent (see Definition 7.1.7) states in $A_2'$ $(q_2)$ and $A_1'$ $(q_1)$ respectively.

We prove that $e'$ satisfies the requirements of Definition 7.1.7, that is, it is a right congruence, too.

1) Obviously, $q_{02}'\ e'\ q_{01}'$.

2) Let $q_2'\ e'\ q_1'$ and let $q_2 \in Q_2$, $q_1 \in Q_1$ with $e(q_2) = q_1$ so that $q_2'$ and $q_1'$ are right-congruent states in $A_2'$ $(q_2)$ and $A_1'$ $(q_1)$ respectively.
Then there exists an arc $\phi'$: $q_2' \rightarrow \theta_2'$ in $A_2'$ $(q_2)$ if and only if there exists an arc $\phi'$: $q_1' \rightarrow \theta_1'$ in $A_1'(q_1)$, where $\theta_2'$ and $\theta_1'$ are right-congruent states in $A_2'$ $(q_2)$ and $A_1'$ $(q_1)$ respectively. Thus there exists an arc $\phi'$: $q_2' \rightarrow \theta_2''$ in $A_2'$ if and only if there exists an arc $\phi'$: $q_1' \rightarrow \theta_1''$ in $A_1'$.
It remains to be shown that $\theta_2''\ e'\ \theta_1''$. If $\theta_1'$ and $\theta_2'$ are both non-terminal states of $M_1'(q_1)$ and $M_2'(q_2)$ then $\theta_2'' = \theta_2'$ and $\theta_2'' = \theta_2'$ hence $\theta_2''\ e'\ \theta_2''$. Otherwise let us assume that $\theta_2'$ is a terminal state in $M_2'(q_2)$ and let $p'$: $q_2 \rightarrow \theta_2'$ and $p'$: $q_1 \rightarrow \theta_1'$ be paths in $A_2'$ $(q_2)$ and $A_1'$ $(q_1)$ respectively (such paths exist since $\theta_2'$ and $\theta_1'$ are right-congruent states). Let also $m \in$ *Memory* be a memory value of $M_2$ that is attainable in $q_2$ (such $m$ exists since $A_2$ is accessible and $\Phi$ is t-complete w.r.t. *Memory*) and let $m' \in h^{-1}(m)$. Since $\Phi'$ is t-complete w.r.t. *Memory*' there exists an input sequence $s'^*$ with $(m', s'^*) \in$ domain $|p'|$. Since $M_2'$ $(q_2)$ is a refining machine it follows that $s'^* \in$ domain $u$ and there exists an arc $\phi_2$: $q_2 \rightarrow \theta_2''$ in $A_2$ with $(m, u(s'^*)) \in$ domain $\phi_1$.
Since $M_1'(q_1)$ is a refining machine w.r.t. $u$, $s'^* \in$ domain $u$ and $(m', s'^*) \in$ domain $|p'|$ it follows that $\theta_1'$ is a terminal state in $M_1'(q_1)$. Also, since $e$ is a finite state machine morphism and $e(q_2) = q_1$ it is easy to see that $m$ is also a memory value of $M_1$ that is attainable in $q_1$. Thus there exists an arc $\phi_1$: $q_1 \rightarrow \theta_1''$ in $A_1$ with $(m, u^*(s'^*)) \in$ domain $\phi_1$. Since $e(q_2) = q_1$ and $M_1$ and $M_2$ are deterministic it follows that $\phi_1 = \phi_2$. Hence $\theta_2''\ e'\ \theta_1''$.

**Lemma** 8.1.2.
Let $M_1$, $M_2$, $M_1'$, $M_2'$ be as above, let $Y \subseteq$ seq(*Input*) be a test set of $M_1$ and $M_2$ and $Y' \subseteq$ seq(*Input*') a set of sequences of *Input*' that refines $Y$ w.r.t. $u$. If $\forall\ s'^* \in Y'$, $f_1'(s'^*) = f_2'(s'^*)$ then there exists a surjective finite state machine morphism, $e$: $A_2 \rightarrow A_1$.

*Proof*:
If $\forall\ s'^* \in Y'$, $f_1'(s'^*) = f_2'(s'^*) \Rightarrow \forall\ s'^* \in Y'$, $v^*(f_1'(s'^*)) = v^*(f_2'(s'^*)) \Rightarrow \forall\ s'^* \in Y'$, $f_1(u^*(s'^*)) = f_2(u^*(s'^*)) \Rightarrow \forall\ s^* \in Y$, $f_1(s^*) = f_2(s^*)$. Hence $A_1$ and the minimal automaton of $A_2$ are isomorphic. Since $A_1$ is minimal and $A_2$ is accessible there exists a surjective morphism, $e$: $A_2 \rightarrow A_1$.

**Definition** 8.1.2.
Let $\{M_1'(q_1) : q_1 \in Q_1\}$ be the refining set of $M_1'$ as above. Then $X_{q1} \subseteq seq(\Phi')$ is called a *distinguishing set* of $A_1'(q_1)$ if for any $\theta_1 \in Q_1$ either $A_1'(q_1)$ and $A_1'(\theta_1)$ are equivalent or $X_{q1}$ distinguishes between $q_1$ and $\theta_1$ as states in $A_1'(q_1)$ and $A_1'(\theta_1)$ respectively.

**Example** 8.1.2.
For the refinement of Example 6.3.3.2 we can take
$X_A$ = {**type_ch1** :: **wrong_name'**}
$X_B$ = {**type_ch1** :: **wrong_psw'**}
$X_C$ = {**type_ch1** :: **type_ch1**:: **new_psw'**}

**Definition** 8.1.3.
Let S be a state cover of $A_1$, $k$ a positive integer and let $t$ : seq($\Phi$) $\rightarrow$ seq(*Input*) be a fundamental test function of $M_1$. Let also

$$P_k = \{ p \in S \bullet (\Phi^k \cup \Phi^{k-1} \cup ... \cup \{< >\}) : \exists\ p \text{ a path in } A_1 \text{ that starts in } q_{01}\}$$

For any $p \in P_k$ let $q_p \in Q_1$ be the end state of $p$ and let $t(p) = s_p^*$. Since $\Phi$ is t-complete w.r.t. *Memory* there exists $m_p \in$ *Memory* with $\pi_2(p(m_0, s_p^*)) = m_p$. Let $s_p^{*'} \in$ seq(*Input'*) with $u^*(s_p^{*'}) = s_p^*$. We will denote $s_p^{*'}$ by $u^{-1}(t(p))$.

Since $M_1'$ is a state refinement of $M_1$ there exists $m_p' \in h^{-1}(m_p)$ and a path $p'$ in $A_1'$ from $q_{01}$ to $q_p$ with $\pi_2(p'(m_0', s_p^{*'})) = m_p'$. Then let $t_p = t_{q_p, m_p'}$ be a test function of $M_1'(q_p)$ w.r.t. $(q_p, m_p')$.

Also let $X_p \subseteq seq(\Phi')$ be a distinguishing set of $A_1'(q_p)$ as defined above.

Then $U_k$ , a set of sequences of *Input'*, defined by:

$$U_k = \bigcup_{p \in P_k} \{ u^{-1}(t(p))\} \bullet (t_p(X_p))$$

will be called a *k-distinguishing set of the refining set*.

**Example** 8.1.3.
Let $M$ and $M'$, be the machines from Example 6.3.3.2. Then a state cover of $M$ is:

$S$ = {< >, **good_name, good_name** :: **good_psw**}

For $k$ = 2, $P_2 = \{p_1, ... p_{12}\}$,
where

$p_1$ = < >, $p_1$: $A \to A$
$p_2$ = **good_name**, $p_2$: $A \to B$
$p_3$ = **wrong_name**, $p_3$: $A \to A$
$p_4$ = **good_name** :: **good_psw**, $p_4$: $A \to C$
$p_5$ = **good_name** :: **wrong_psw**, $p_5$: $A \to A$
$p_6$ = **wrong_name** :: **good_name**, $p_6$: $A \to B$
$p_7$ = **wrong_name** :: **wrong_name**, $p_7$: $A \to A$
$p_8$ = **good_name** :: **good_psw** :: **new_psw**, $p_8$: $A \to A$
$p_9$ = **good_name** :: **wrong_psw** :: **good_name**, $p_9$: $A \to B$
$p_{10}$ = **good_name** :: **wrong_psw** :: **wrong_name**, $p_{10}$: $A \to A$
$p_{11}$ = **good_name** :: **good_psw** :: **new_psw** :: **good_name**, $p_{11}$: $A \to B$
$p_{12}$ = **good_name** :: **good_psw** :: **new_psw** :: **wrong_name**, $p_{12}$: $A \to A$

We will assume that ‘01’ is a valid username and ‘1’ is not a valid username,
i.e. ‘01’ $\in$ domain $\text{map}_0$ and ‘1’ $\notin$ domain $\text{map}_0$ and that ‘11’ is the password for ‘01’, i.e. map(‘01’) = ‘11’.

Then we can take $s_i$*, $s_i$*', $m_i$ and $m_i$', i = 1...12, as follows:
$s_1$* = < >
$s_2$* = *‘01’*
$s_3$* = *‘1’*
$s_4$* = *‘01’ ‘11’*
$s_5$* = *‘01’ ‘0’*
$s_6$* = *‘1’ ‘01’*
$s_7$* = *‘1’ ‘1’*
$s_8$* = *‘01’ ‘11’ ‘00’*
$s_9$* = *‘01’ ‘0’ ‘01’*
$s_{10}$* = *‘01’ ‘0’ ‘1’*
$s_{11}$* = *‘01’ ‘11’ ‘00’ ‘01’*
$s_{12}$* = *‘01’ ‘11’ ‘00’ ‘1’*

$s_1$*' = < >
$s_2$*' = 0 :: 1 :: *enter*
$s_3$*' = 1 :: *enter*
$s_4$*' = 0 :: 1 :: *enter* :: 1 :: 1 :: *enter*
$s_5$*' = 0 :: 1 :: *enter* :: 0 :: *enter*
$s_6$*' = 1 :: *enter* :: 0 :: 1 :: *enter*
$s_7$*' = 1 :: *enter* :: 1 :: *enter*
$s_8$*' = 0 :: 1 :: *enter* :: 1 :: 1 :: *enter* :: 0 :: 0 :: *enter*
$s_9$*' = 0 :: 1 :: *enter* :: 0 :: *enter* :: 0 :: 1 :: *enter*
$s_{10}$*' = 0 :: 1 :: *enter* :: 0 :: *enter* :: 1 :: *enter*
$s_{11}$*' = 0 :: 1 :: *enter* :: 1 :: 1 :: *enter* :: 0 :: 0 :: *enter* :: 0 :: 1 :: *enter*
$s_{12}$*' = 0 :: 1 :: *enter* :: 1 :: 1 :: *enter* :: 0 :: 0 :: *enter* :: 1 :: *enter*

$m_1 = m_3 = m_7 = (\text{map}_0, \text{‘’})$
$m_2 = m_4 = m_5 = m_6 = m_9 = m_{10} = (\text{map}_0, \text{‘01’})$
$m_8 = m_{11} = m_{12} = (\text{new\_map}, \text{‘01’})$
where $\text{new\_map} = \text{map}_0 \oplus (\text{‘01’} \rightarrow \text{‘00’})$

$m_1' = m_3' = m_7' = ((\text{map}_0, \text{‘’}), < >)$
$m_2' = m_4' = m_5' = m_6' = m_9' = m_{10}' = ((\text{map}_0, \text{‘01’}), < >)$
$m_8' = m_{11}' = m_{12}' = ((\text{new\_map}, \text{‘01’}), < >)$

If $X_A$, $X_B$ and $X_C$ are those from Example 8.1.2 then we can take:
$U_2 = \{s_1^{*'} :: t_A^{*'}, s_2^{*'} :: t_B^{*'}, s_3^{*'} :: t_A^{*'}, s_4^{*'} :: t_C^{*'}, s_5^{*'} :: t_A^{*'}, s_6^{*'} :: t_B^{*'}, s_7^{*'} :: t_A^{*'}, s_8^{*'} :: t_A^{*'},$
$s_9^{*'} :: t_B^{*'}, s_{10}^{*'} :: t_A^{*'}, s_{11}^{*'} :: t_B^{*'}, s_{12}^{*'} :: t_A^{*'}\}$

where $t_A^{*'} = 1 :: \text{enter}$, $t_B^{*'} = 0 :: \text{enter}$, $t_C^{*'} = 0 :: 0 :: \text{enter}$

**Theorem** 8.1.1.
Let $M_1$, $M_2$, $M_1'$, $M_2'$ be as in Definition 8.1.3 above. Let $Y \subseteq \text{seq}(Input)$ be a test set of $M_1$ and $M_2$, $Y' \subseteq \text{seq}(Input')$ a set that refines $Y$ w.r.t. $u$ and $U_k$ a $k$-distinguishing set of the refining set of $M_1'$, where $k = \text{Card}(Q_2) - \text{Card}(Q_1)$ is the difference between the number of states of $M_2$ and the number of states of $M_1$ and let $Y'' = Y' \cup U_k$ . If $\forall\, s'^* \in Y''$, $f_1'(s'^*) = f_2'(s'^*)$ then $A_1'$ and $A_2'$ are equivalent.

*Proof*:
From the lemma it follows that there exists $e$: $A_2 \rightarrow A_1$ a surjective finite state machine morphism. Then the set $P_k$ defined as in Definition 8.1.3 is a state cover of $A_2$. Indeed since $e$: $A_2 \rightarrow A_1$ is a finite state morphism at least $\text{Card}(Q_1)$ states of $A_2$ will be accessed by a path in S. Also $\forall\, i = 1,...(k\text{-}1)$ the set $P_{i+1}$ will access at least one of the states that has not been accessed by $P_i$.

Thus for any $q_2 \in Q_2$ there exists $p \in P_k$ so that $p$: $q_{02} \rightarrow q_2$ is a path in $M_2$ and $p$: $q_{01} \rightarrow e(q_2)$ is a path in $M_1$. If $s_p^*$, $s_p^{*'}$, $m_p$ and $m_p'$ are those from Definition 8.1.3 then there exists $p'$: $q_{01} \rightarrow e(q_2)$ a path in $M_1'$ with $\pi_2(p'(m_0', s_p^{*'})) = m_p'$. Since $M_2'$ is a state refinement of $M_2$ there exists $p''$: $q_{02} \rightarrow q_2$ a path in $M_2'$ and $m_p'' \in h^{-1}(m_p)$ with $\pi_2(p''(m_0', s_p^{*'})) = m_p''$. From the output-distinguishability of $\Phi'$ it follows that $p'' = p'$ and $m_p'' = m_p'$ .

Since $\Phi'$ is output-distinguishable the application of $t_{q_p}(X_p)$ in $q_2$ and $e(q_2)$ respectively with initial memory $m_p'$ will ensure that $A_2'(q_2)$ and $A_1'(q_1)$ are equivalent. From Lemma 8.1.1. it follows that $A_1'$ and $A_2'$ are equivalent.

## 8.2.The refinement testing method.

The previous theorem can be used as the theoretical basis of a method for testing stream X-machine specifications constructed through a process of state refinement. The method will use a two phase approach in which the refining machines are implemented and tested first and then the entire system is tested for integration.

*Phase 1. Testing the refining machines.*
These will be implemented as separate subprograms or pieces of code that can be tested in isolation from the rest of the system. Each such *subimplementation* will communicate with the rest of the system through one or more "transfer variables" whose values will correspond to the terminal states of the refining machine. Thus, the testing procedure will not only have to ensure that the input/output behaviour of the refining machine coincides with that of its implementation but also that each transfer variable identifies correctly its corresponding terminal state. Therefore we need to adapt our refining machines so that they possess the transfer variable linking mechanism. This implies that any component technology must also incorporate a suitable mechanism for specifying transfer variables. The process can be described formally by an augmented refining machine as defined next.

**Definition** 8.2.1.
Let $M_1'(q_1) = (Input', Output', R_{q1}, Memory', \Phi', \mathbf{F}_{q1}, q_1, m_{q1}', T_{q1})$ be the refining machine in $q_1$, $T_{q1} = \{t_1, ..., t_m\}$ the set of terminal states and let $in_{q1} \notin Input'$ and
$Output_{Tq1} = \{out_{t1}, ..., out_{tm}\}$ a set with $Output_{Tq1} \cap Output' = \varnothing$. Then a stream X-machine defined by:
$M_{1A}'(q_1) = (Input'_A, Output'_A, R_{q1}, Memory', \Phi'_A, \mathbf{F}_{q1A}, q_1, m_{q1}', T_{q1})$ where:
1. $Input'_A = Input' \cup \{in_{q1}\}$.
2. $Output'_A = Output' \cup Output_{Tq1}$.
3. $\Phi'_A = \Phi' \cup \Psi_{q1}'$, with $\Psi_{q1}' = \{\psi_{t1}', ..., \psi_{tm}'\}$
such that, for any $t \in T_{q1}$, $\psi_t'$ is defined by
$\psi_t'(m, in_{q1}) = (out_t, m)$, $m \in Memory$
4. $\mathbf{F}_{q1A}: R_{q1} \times \Phi'_A \to R_{q1}$ is defined by

$$\mathbf{F}_{q1A}(q', \phi') = \begin{cases} \mathbf{F}_{q1}(q', \phi'), \text{ if } q' \in R_{q1} \text{ and } \phi' \in \Phi' \\ q', \text{ if } \exists\, j = 1...m \text{ with } q' = t_j \text{ and } \phi' = \psi_{tj}' \\ \perp, \text{ otherwise} \end{cases}$$

is called the *augmented refining machine* in $q_1$ .

In other words each terminal state $t$ will be identified in the augmented machine by an extra output symbol $out_t$ produced by an extra processing function $\psi_t$ that labels a loop from $t$. These loops will be triggered by an extra input $in_{q1}$. The extra outputs will correspond to the values of transfer variables used by the implementation. The extra input $in_{q1}$ does not necessarily exist in the implementation; it will correspond to a means of identifying the value of the transfer variables for each terminal state. The augmented refining machines of Example 6.3.5 may be found in figure 8.2.

If an augmented machine $M_{1A}'(q_1)$ is tested against its corresponding implementation using

the SXMT method and the implementation passes the required tests then this can be modelled as a stream X-machine $M_2'(q_2)$ whose associated automaton is equivalent to that of $M_1'(q_1)$ and whose terminal states - that is the transfer variables of the implementation - will match the terminal states of $M_1'(q_1)$.

Obviously, the SXMT method will require that the basic processing functions $\Phi'$ satisfy the "design for testing conditions" and are implemented correctly.

*Phase 2. Testing the system for integration*

The system will be composed of the above implementations that communicate through the transfer variables. If the first phase of our testing method has been completed then it easy to see that the entire system can be modelled by a stream X-machine $M_2'$ with type $\Phi'$ that is a state refinement of a machine with the same type $\Phi$ as $M_1$. Thus Theorem 8.1.1. can be used to generate a test set $Y'' = Y' \cup U_k$ whose application guarantees that the system is functionally equivalent to its specification. Recall that if W is a characterisation set of $A_1$ then

$Y = t\,(S \bullet (\Phi^{k+1} \cup \Phi^k \cup ... \cup \Phi) \bullet W)$ is a test set of $M_1$ and $M_2$, thus $Y' = u^{-1}*(Y)$.

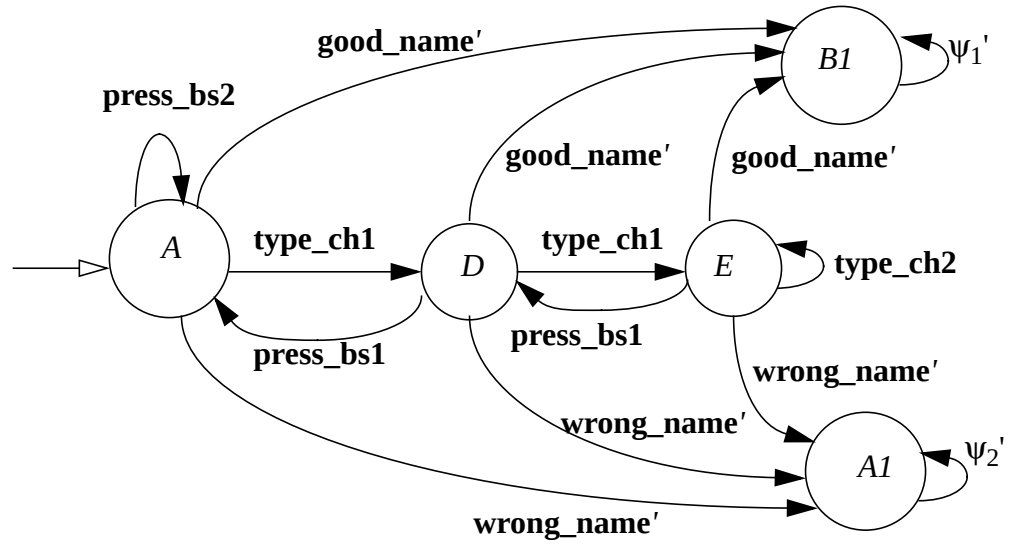
*Complexity and upper bounds of the test set*.

If the complexity of $u^{-1}$is $C(u^{-1})$, the complexity of each $\phi \in \Phi$ is at most $C$ and the complexity of each procedure used to determine a distinguishing set of a refining machine is at most $C_D$ then the complexity of the algorithm that generates the test set $Y''$ is proportional to

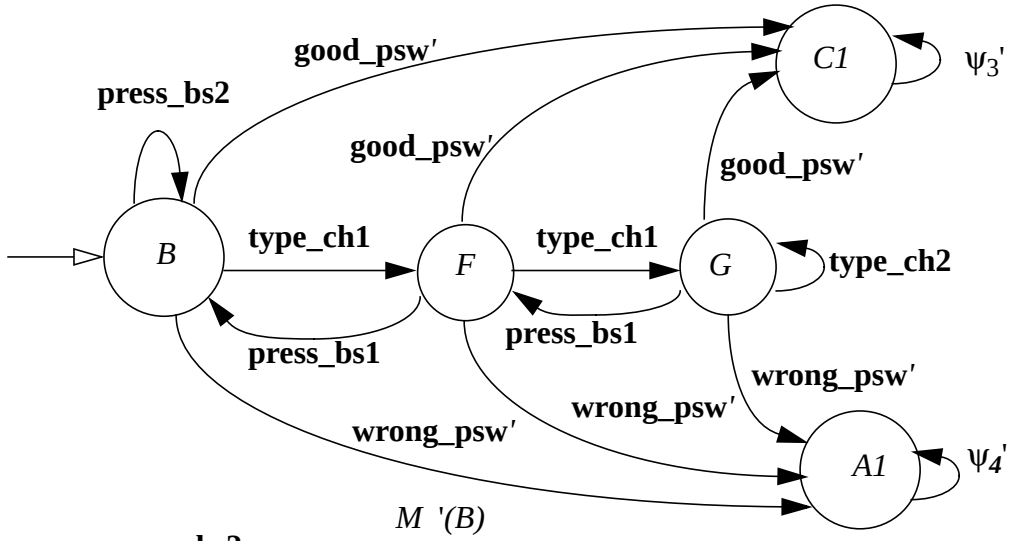
$$C(u^{-1}) \,.\, C \,.\, p \,.\, r^{k+1} \,.\, n^2 \,.\, (n+2k) + r^k \,.\, n \,.\, C_D$$

where $n$ = Card($Q_1$), $k$ = Card($Q_2$) - Card($Q_1$), $p$ = Card(*Input*) and $r$ = card($\Phi$).

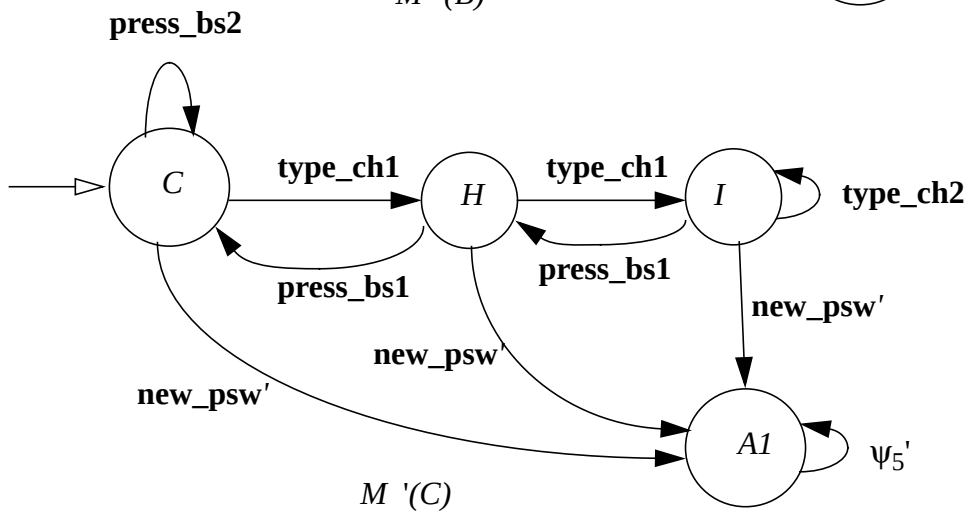
If all values of $u^{-1}$are sequences of at most $l$ elements and the length of any sequence used to distinguish between any two refinement modules is at most $l$, then the upper bounds for the number of test sequences required and the total length of the test set are approximately $l \,.\, n^2 \,.\, r^{k+1}$ and $l \,.\, (n+k) \,.\, r^{k+1}$ respectively.




*M ′(A)*




*M ′(B)*




*M ′(C)*

**Figure 8.1.**

*Design for testing requirements*

As previously discussed, the “design for test conditions” (i.e. the t-completeness and output-distinguishability of the types $\Phi$ and $\Phi'$) can be introduced into the specification through an augmentation of the input and/or output alphabets. However, in this case, particular care is needed to ensure that the augmented machines will still be in a state refinement relation. For example, it is easy to see that $\Phi$ and $\Phi'$ of Example 6.3.3.2 are output-distinguishable but not t-complete. However, the t-completeness of $\Phi$ and $\Phi'$ can be achieved through an extension of their input alphabets:

$In_e = In \cup \{in_1, in_2\}$, where $In \cap \{in_1, in_2\} = \varnothing$

$In_e' = In' \cup \{in_1', in_2'\}$, where $In' \cap \{in_1', in_2'\} = \varnothing$

These extra inputs will be used to extend the processing functions in $\Phi$ and $\Phi'$ as follows. We only show the effect of the extra inputs, the processing functions remain unchanged elsewhere. Let *map* $\in$ MAPS, *name*, *str* $\in$ $\text{WORDS}_2$,

$\mathbf{good\_name_a}((map, name), in_1) = ((msg1, str), (map, name))$

$\mathbf{wrong\_name_a}((map, name), in_2) = ((msg2, str), (map, name))$

$\mathbf{good\_psw_a}((map, name), in_1) = ((msg3, str), (map, name))$

$\mathbf{wrong\_psw_a}((map, name), in_2) = ((msg4, str), (map, name))$

$\mathbf{good\_name'_a}(((map, name), str), in_1') = (msg1, ((map, name), < >))$

$\mathbf{wrong\_name'_a}(((map, name), str), in_2') = (msg2, ((map, name), < >))$

$\mathbf{good\_psw'_a}(((map, name), str), in_1') = (msg3, ((map, name), < >))$

$\mathbf{wrong\_psw'_a}(((map, name), str), in_2') = (msg4, ((map, name), < >))$

It is easy to see that the augmented types are both t-complete and output-distinguishable. The only thing that remains to be done is to extend the definition of the input covering (we will call this $u_{2e}$) alphabet by:

$u_{2e}(a^* :: in_1') = in_1$
$u_{2e}(a^* :: in_2') = in_2$
$\forall\ a^* \in \text{seq}(\text{CHARS} \cup \{backspace\})$ with $a^* :: enter \in \text{domain } u_2$.