# TESTE DE IMPLEMENTAÇÕES DE PROTOCOLOS.

Newton Alberto de Castilho Lages.
Eduardo Paoliello.

Universidade Federal de Minas Gerais - UFMG.
Departamento de Ciência da Computação - ICEx.
Av. Antonio Carlos, 6627 Tel. (031)443-4088.
Belo Horizonte - MG -

RESUMO

A definição das sequências de teste a serem aplicadas em uma implementação de um protocolo constitui-se num objeto de pesquisa intensa por parte da comunidade científica. Diversas técnicas vêm sendo utilizadas para a seleção e geração destes testes [Sarikaya,87]. O objetivo deste trabalho é apresentar uma metodologia de geração de sequências de teste para protocolos. Esta metodologia foi introduzida por [Chow,78] e se aplica a qualquer software que possa ser modelado como uma máquina de estados finita. Apenas a estrutura de controle pode ser testada. Um protocolo de transporte é utilizado como exemplo para demonstrar a aplicação desta metodologia.

## 1. Introdução.

Recentemente, bastante atenção vem sendo dada ao teste de implementações de protocolos. Dentre os trabalhos pioneiros nesta área, podemos citar o projeto inglês NPL [Rayner,82], o projeto francês RHIN [Rafiq,86] e o americano da NBS (National Bureal of Standards) [Linn & Nightingale,83a]. A elaboração de sequências de teste efetivas constitui uma das atividades mais importantes no teste de protocolos de comunicação. Estas sequências devem ser elaboradas de forma a endereçar aspectos relevantes dos serviços e dos mecanismos do protocolo, para verificar se eles são providos corretamente pela implementação sob teste (IST) [Giozza et al.,86b].

O objetivo deste trabalho é projetar e implementar um gerador automático de sequências de teste, a partir de uma máquina de estados finita. Um protocolo de transporte padrão ISO, classe 0, é utilizado como exemplo na demonstração do uso desta metodologia.

O ítem 2 apresenta uma especificação de um protocolo de transporte da ISO, classe 0. No ítem 3, é apresentado o método proposto para a geração de sequências de teste efetivas. No ítem 4 o projeto do gerador é apresentado. E no ítem 5 são apresentadas algumas conclusões da utilização deste método.

## 2. Especificação de um Protocolo de Transporte.

O objetivo da camada de transporte no modelo OSI é prover a transferência de dados fim-a-fim entre os diversos usuários da rede. Em função das diferentes qualidades do serviço de rede, a ISO define cinco classes de protocolo de transporte.

Um protocolo de transporte classe 0, modelado na forma de uma máquina de estados finita, será utilizado como exemplo na descrição da metodologia de geração de sequências de testes. A figura 1 ilustra a especificação deste protocolo e a figura 2 contém um detalhamento dos estados, primitivas de serviço, predicados de habilitação, e TPDUs contidas nesta especificação.

Existem 12 entradas e 12 saídas definidas para este protocolo:

Entradas: TCONreq, DR, ER, DT, CC, CR, TDISreq, NDISind, NRSTind, TDTreq, TCONresp e NCONconf.

Saídas: -, TCONind, TCONconf, TDTind, TDISind, NDISreq, NCONreq, CR, DT, CC, ER e DR.

Normalmente, esta metodologia é apresentada com uma simplificação na máquina de estados finita, que é a desconsideração dos predicados de habilitação nos diversos estados da máquina. Porém, consideramos esta simplificação não obrigatória, desde que os predicados de habilitação sejam incluídos nos eventos de entrada. Por exemplo, baseando-se na especificação da figura 1, a entrada TCONreq é desdobrada em quatro eventos de entrada, que são TCONreq.[P0], TCONreq.[P2], TCONreq.[P3] e TCONreq.[P4]. Esta proposta torna a aplicação da metodologia de geração de testes um pouco mais abrangente. Contudo, esta abordagem só pode ser aplicada em arquiteturas de teste locais, onde as condições do ambiente que cercam a Implementação Sob Teste (IST) são controláveis. Se a arquitetura de testes for distribuída, a simplificação citada acima torna-se realmente necessária.

Deve-se observar também, que a especifição do protocolo de transporte da figura 1 não está definida completamente, ou seja, nem todas as entradas estão especificadas para todos os estados. A fim de tornar o protocolo mais fácil de ser testado, pode-se complementar a especificação. O ítem 4.3 apresenta duas propostas diferentes para se complementar uma especificação, permitindo a realização de testes mais completos.

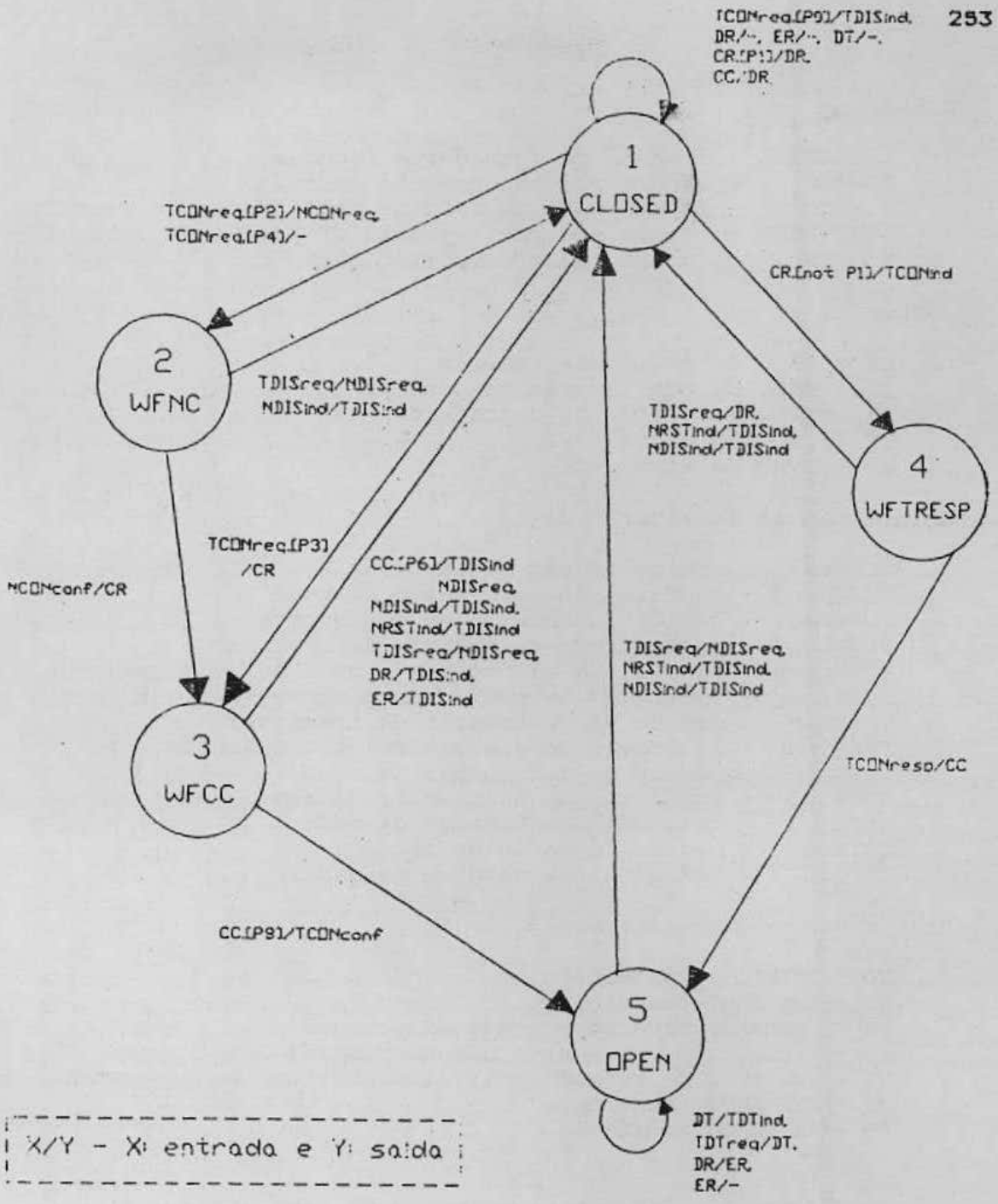


**Figura 1 - Especificação do Protocolo de Transporte Classe 0**

Estados:

1 CLOSED - conexão de transporte fechada
2 WFNC - espera por uma conexão de rede
3 WFCC - espera por uma TPDU de confirmação de conexão
4 WFTRESP - espera por uma resposta de pedido de conexão
5 OPEN - conexão de transporte aberta

TPDUs:

CR - TPDU de pedido de conexão
CC - TPDU de confirmação de conexão
DR - TPDU de pedido de desconexão
DT - TPDU de dados
ER - TPDU de erro

Primitivas de Serviço:

NCONreq - pedido de conexão de rede
NCONconf - confirmação de conexão de rede
TCONreq - pedido de conexão de transporte
TCONconf - confirmação de conexão de transporte
TCONresp - resposta a um pedido de conexão de transporte
TCONind - indicação de pedido de conexão de transporte
TDISreq - pedido de desconexão de transporte
TDISind - indicação de desconexão de transporte
NDISreq - pedido de desconexão de rede
NDISind - indicação de desconexão de rede
NRSTind - indicação de "reset" de rede
TDTreq - pedido de envio de dados
TDTind - indicação de chegada de dados

Predicados de Habilitação:

P0 - TCONreq inaceitável
P1 - CR TPDU inaceitável
P2 - nenhuma conexão de rede disponível
P3 - conexão de rede aberta e disponível
P4 - conexão de rede disponível e abertura em progresso
P6 - CC TPDU inaceitável
P8 - CC TPDU aceitável

Figura 2 - Estados, predicados de habilitação, primitivas de serviço, e TPDUs da especificação do Protocolo de Transporte Classe 0

# 3. O Método W na Geração de Sequências de Teste.

## 3.1. Introdução.

O método W foi inicialmente proposto por [Chow,78]. O objetivo deste método é testar a correção de programas especificados como máquinas de estados finitas. Os resultados dos testes, obtidos da implementação, são confrontados com a especificação. Considerando-se que apenas a estrutura de controle é testada, o método W é uma estratégia de seleção de dados de teste válida e confiável, pois detecta todos os erros da função de saída e da função de próximo estado.

Para a utilização do método W, supõe-se que a máquina:

- encontra-se em sua forma mínima, isto é, a máquina não possui estados equivalentes;
- possui um conjunto de entrada bem definido;
- possui um número máximo de estados conhecidos;
- começa em um estado inicial fixo;
- é fortemente conectada.

Além destas características, assume-se que a máquina é do tipo Mealy. Isto significa que o valor da saída é uma função dependente do estado da máquina e do valor da entrada.

As sequências de teste são obtidas através das seguintes etapas, descritas nos itens subsequentes:

- definição de um conjunto de entradas (conjunto W) que identifica, de forma única, cada estado da máquina (ítem 3.2);
- geração de uma árvore de teste, a partir da máquina de estados finita, e obtenção de todos os caminhamentos parciais da árvore de teste (ítem 3.3);
- geração de um conjunto Z, para suportar estados adicionais da implementação (ítem 3.4);
- concatenação do conjunto Z com o conjunto P, para a obtenção das sequências de teste.

## 3.2. Construção do conjunto W.

O conjunto W é um conjunto de entradas que podem identificar os estados da máquina pelas respostas a estas sequências de entradas (sequências de identificação) [Kou,87]. Definido de outra forma, um conjunto W, ou conjunto de caracterização, consiste de sequências de entradas que podem distinguir os comportamentos de cada par de estados do autômato mínimo [Chow,78]. Estas sequências de caracterização não são únicas para a máquina que está sendo testada, ou seja, diferentes sequências podem ser aplicadas para a identificação dos estados da MEF.

A obtenção de um conjunto W não é uma tarefa simples para todos os casos. Muitas vezes, o estado da máquina em um ponto não pode ser determinado meramente observando as respostas da máquina naquele ponto. Ao invés disso, é necessário coletar informações de diversos pontos da máquina simultaneamente, e observar as respostas do estado em questão a diferentes sequências de entrada [Kohavi,78]. As técnicas utilizadas para a geração do conjunto W baseiam-se em conceitos de equivalência e identificação de estados de máquinas de estados finitas.

Alguns métodos de obtenção do conjunto W baseiam-se no particionamento dos estados da máquina. Um destes métodos, descrito em [Gill,62], baseia-se na construção de tabelas, que representam diversos níveis de particionamento da máquina de estados. Estas tabelas, denominadas tabelas Pk, são construídas até que todos os estados estejam agrupados em classes equivalentes. Para uma máquina mínima, existem pelo menos n-1 tabelas Pk, onde n representa o número de estados.

A identificação dos diversos estados da máquina é obtida pela utilização de um algoritmo nas tabelas Pk. Ele seleciona uma sequência de entradas nos diversos níveis de particionamento da máquina, de tal forma que a saída final para aplicação desta sequência é diferente para cada par de estados.

Outras técnicas para a construção de conjuntos de caracterização, também baseadas nas partições da máquina de estados, estão disponíveis [Kohavi,78].

Para o protocolo de transporte classe 0, alguns conjuntos de caracterização podem ser obtidos. Os conjuntos {DR,TDISreq,NCONconf} e {DR,TCONresp,NCONconf,NDISind} são conjuntos W, pois a aplicação de alguns de seus elementos identificam de forma única os estados definidos na especificação do protocolo de transporte. A figura 3 apresenta um quadro de respostas para cada estado da máquina, com a aplicação de {DR,TDISreq,NCONconf}.

Toda máquina completamente especificada e mínima possui um conjunto W. No entanto, este método pode ser aplicado em máquinas incompletamente especificadas, desde que exista um conjunto W.

A inclusão de predicados de habilitação nos eventos de entrada pode dificultar a obtenção de um conjunto W mínimo. Isto se deve ao fato de que os predicados de habilitação só se aplicam a alguns estados da máquina e não a todos. Considerando-se que, apenas parte do conjunto W é aplicado para cada estado, os eventos de entrada que possuirem predicados associados, podem ser incluídos no conjunto W, desde que se apliquem a estados onde os predicados de habilitação possuam um significado associado.

| ENTRADA/ ESTADO | DR | TDISreq | NCONconf |
|---|---|---|---|
| | | SAIDA | |
| 1 CLOSED | — | — | — |
| 2 WFNC | — | NDISreq | CR |
| 3 WFCC | TDISind | NDISreq | — |
| 4 WFTRESP | — | DR | — |
| 5 OPEN | ER | NDISreq | — |

ENTRADA = <DR, TDISreq, NCONconf>

Figura 3 - Respostas ao conjunto de caracterização (DR,TDISreq,NCONconf)

3.3. Construção do conjunto P.

Para se obter o conjunto P, deve-se construir uma árvore de teste. A árvore de teste é definida de tal forma que os nós representam os estados da MEF e as arestas representam as transições de entrada. P é o conjunto contendo todos os caminhamentos parciais na árvore de teste, inclusive a sequência vazia. Um caminhamento parcial é uma sequência de arestas (transições de entrada) que se inicia na raiz da árvore e termina em cada nó terminal ou não terminal.

O conjunto P representa, portanto, qualquer conjunto de estados tal que, para cada transição de um estado Si para um estado Sj na entrada de x, existem sequências de entrada p e px em P, onde p força a máquina para o estado Si, a partir de seu estado inicial. Considerando-se inicialmente uma máquina de estados finita M, uma árvore de teste T é construída da seguinte forma [Chow,78]:

a. Nomear a raiz de T com o estado inicial de M. Este é o nível 1 de T.
b. Supor que T já tenha sido construído para um nível k. O nível (k+1) é construído examinando-se os nós do nível k, da esquerda para a direita. Um nó no nível k termina se o seu nome é o mesmo que o de um nó não terminal em um nível j, j <= k. Se na entrada x, a máquina M vai do estado Si para o estado Sj, conecta-se uma aresta e um nó sucessor para o nó Si em T. A aresta e o nó sucessor são nomeados x e Sj, respectivamente.

O processo termina quando todos os estados da máquina se encontram na árvore de teste como nós não terminais. É importante notar, que o número de nós não terminais corresponde ao número de estados da máquina. Dependendo da ordem em que os nós sucessores são colocados, diferentes árvores de teste podem ser geradas. A árvore de teste gerada é ilustrada na figura 4.

### 3.4. Construção do conjunto Z.

O objetivo de se construir o conjunto Z é possibilitar o teste de implementações cujo número de estados excede o definido na especificação. A fórmula que define o conjunto Z é mostrada na figura 5.

Normalmente, o número de estados da implementação é maior que o número de estados da especificação ($m >= n$). Se este número for igual ($m = n$), Z será reduzido ao conjunto W ($Z = W$).

As sequências de testes são obtidas pela concatenação dos conjuntos P e Z. Após a aplicação de P, a máquina é exercitada com o conjunto Z, a fim de reconhecer o estado em que a máquina se encontra e, se $m > n$, explorar estados adicionais criados pela implementação do protocolo.

É importante observar que o método W verifica se o autômato implementado é P.Z equivalente ao autômato definido na especificação. Durante o processo de testes, compara-se todas as sequências de saída geradas, não por todas as sequências de entrada mas, apenas, pelo conjunto P.Z.

## 4. Projeto de um Gerador de Sequências de Teste.

### 4.1. Introdução.

Um teste é uma sequência de interações cujo objetivo é verificar uma função da IST. Cada teste é uma unidade independente, responsável pelo completo gerenciamento de uma conexão com a IST, utilizando informações coletadas pelos testes anteriores e, possivelmente, suprimindo alguma informação para os testes seguintes [Bochmann et al.,83].

Baseado neste conceito e na metodologia apresentada no ítem anterior, um gerador de sequências de teste foi projetado e implementado. Cada sequência de teste começa e termina no estado inicial da máquina. As sequências de teste geradas baseiam-se unicamente na especificação do protocolo, definido como uma máquina de estados finita.

Uma simplificação usualmente adotada na metodologia descrita anteriormente, e imposta neste projeto, é a consideração de que o número de estados da especificação (n) é igual ao número de estados da implementação (m), ou seja, (m = n), o que torna (Z = W) (item 3.4).

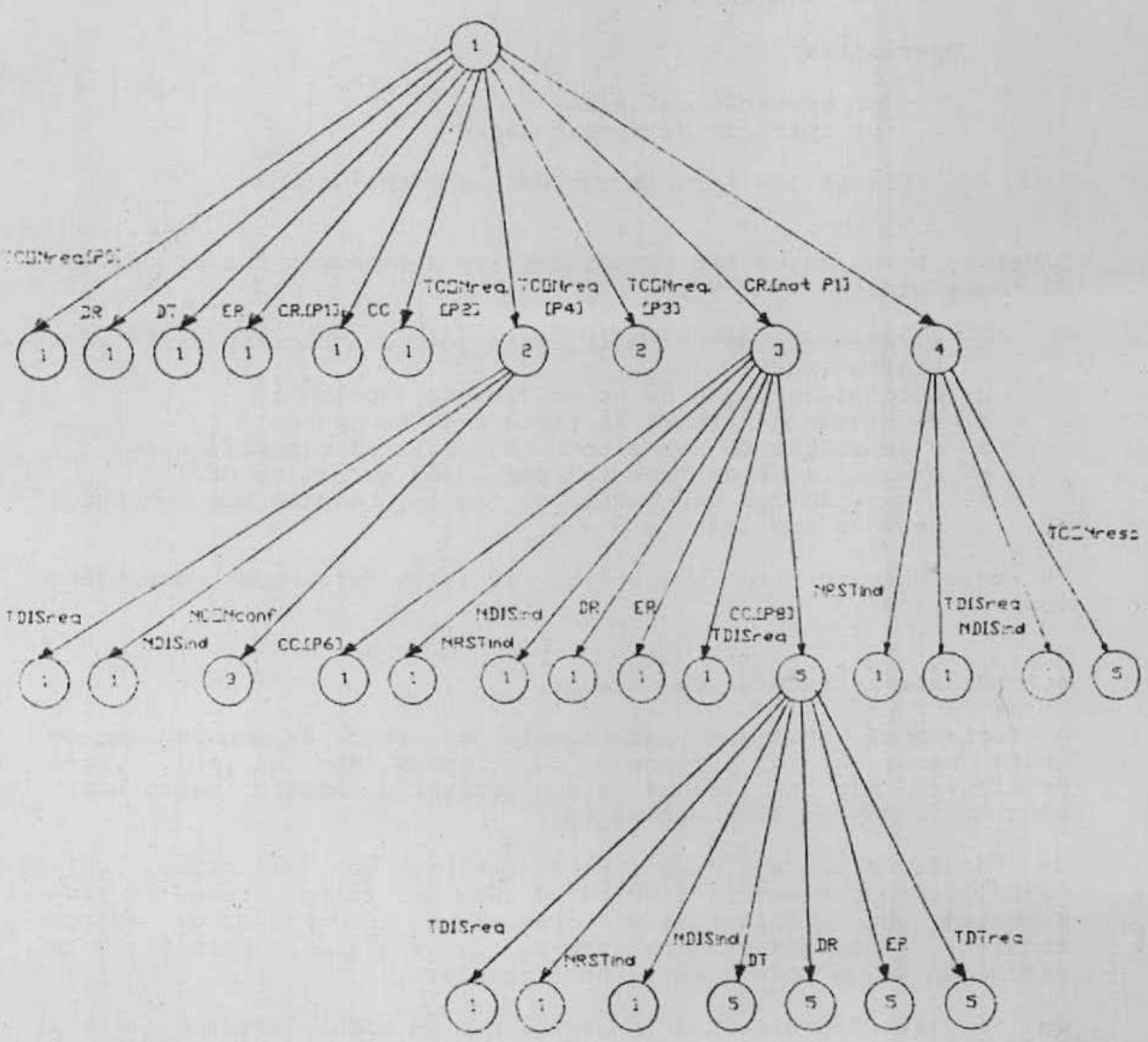


Figura 4 - Árvore de Teste do Protocolo de Transporte Classe 0

Fórmula:

$$Z = W \cup X.W \cup X^{2}.W \ldots \cup X^{(m-n)}.W$$

Variáveis:

m: número de estados da implementação
n: número de estados da especificação
X: conjunto de entrada
W: conjunto W

Operadores:

u: operação de união
.: operação de concatenação

Figura 5 - Fórmula que define o Conjunto Z

Diversas fases devem ser executadas até a obtenção das sequências de teste finais:

a. a definição da especificação como uma Máquina de Estados Finita (MEF);
b. a complementação da especificação (opcional);
c. a geração da árvore de teste e do conjunto P;
d. a definição do conjunto W, a partir da especificação;
e. a elaboração de "resets" para cada estado da MEF;
f. a geração das sequências de testes, baseado nos conjuntos gerados nos ítens c,d e e.

Os ítens subsequentes apresentam, de forma detalhada, o projeto deste gerador.

## 4.2. Modelo Estrutural do Gerador.

A ferramenta utilizada para modelar o gerador de sequências de teste baseia-se na proposta de [Campos et al.,80]. Três primitivas são utilizadas para a criação do modelo estrutural: módulos, soquetes e interconexões.

Um módulo é uma entidade com fronteiras bem definidas, cuja comunicação com outros módulos só pode ser feita através de seus soquetes. Um módulo pode ser composto de um conjunto de outros módulos, interconectados através de soquetes, permitindo a definição de um modelo estrutural completo.

Um soquete representa a interface que um módulo oferece ao seu meio ambiente. Seu nome é conhecido dentro e fora do módulo, o que permite sua utilização por outros módulos.

Uma interconexão é uma primitiva usada para ligar dois ou mais soquetes e, consequentemente, dois ou mais módulos.

Um módulo pode representar, indistintamente, dois tipos de recursos. O módulo é ativo, quando ele provoca alterações no estado do sistema, e é composto por um conjunto de procedimentos cujos nomes estão associados aos soquetes. O módulo é passivo, quando representa uma estrutura de dados, manipulável através de suas operações básicas que são seus soquetes.

O modelo estrutural do gerador de sequências de teste é mostrado na figura 6. O nome do soquete escrito fora do contorno do módulo indica que o respectivo módulo importa aquele recurso e, o nome do soquete escrito internamente ao contorno do módulo indica que o recurso é exportado pelo módulo.

## 4.3. Definição da Especificação como uma MEF.

Basicamente, toda a composição de uma sequência de teste é baseada na máquina de estados finita associada ao protocolo.

O módulo CRIA_ESP permite, de forma interativa, a definição de uma especificação como uma máquina de estados finita. Durante a elaboração da especificação, diversas verificações são realizadas a fim de se obter uma especificação robusta e consistente. Uma exigência básica para que estas consistências possam ser feitas é que as transições devem ser informadas em ordem crescente de estado inicial. Por exemplo, todas as transições partindo do estado 1 são informadas, a seguir todas do estado 2, etc. As verificações feitas por este módulo são:

- se não existem entradas ou saídas repetidas ou nulas;
- se o número de transições é compatível como o número de estados;
- se uma entrada pertence a apenas uma transição partindo de um mesmo estado;
- se existem entradas repetidas dentro de uma mesma transição;
- se o número de entradas por transição não excede o número máximo de entradas da MEF;
- se as entradas em cada transição pertencem ao conjunto de entradas da MEF;
- se o número de saídas por transição não excede o número máximo de saídas da MEF;
- se as saídas em cada transição pertencem ao conjunto de saídas da MEF;
- se uma transição é única na MEF.

O módulo CRIA_ESP possui uma interface com os módulos CONSOLE, CONTROLE e ESPECIFIC. Este último é um módulo passivo, criado pelo módulo ativo CRIA_ESP, a partir dos dados gerados no módulo CONSOLE.

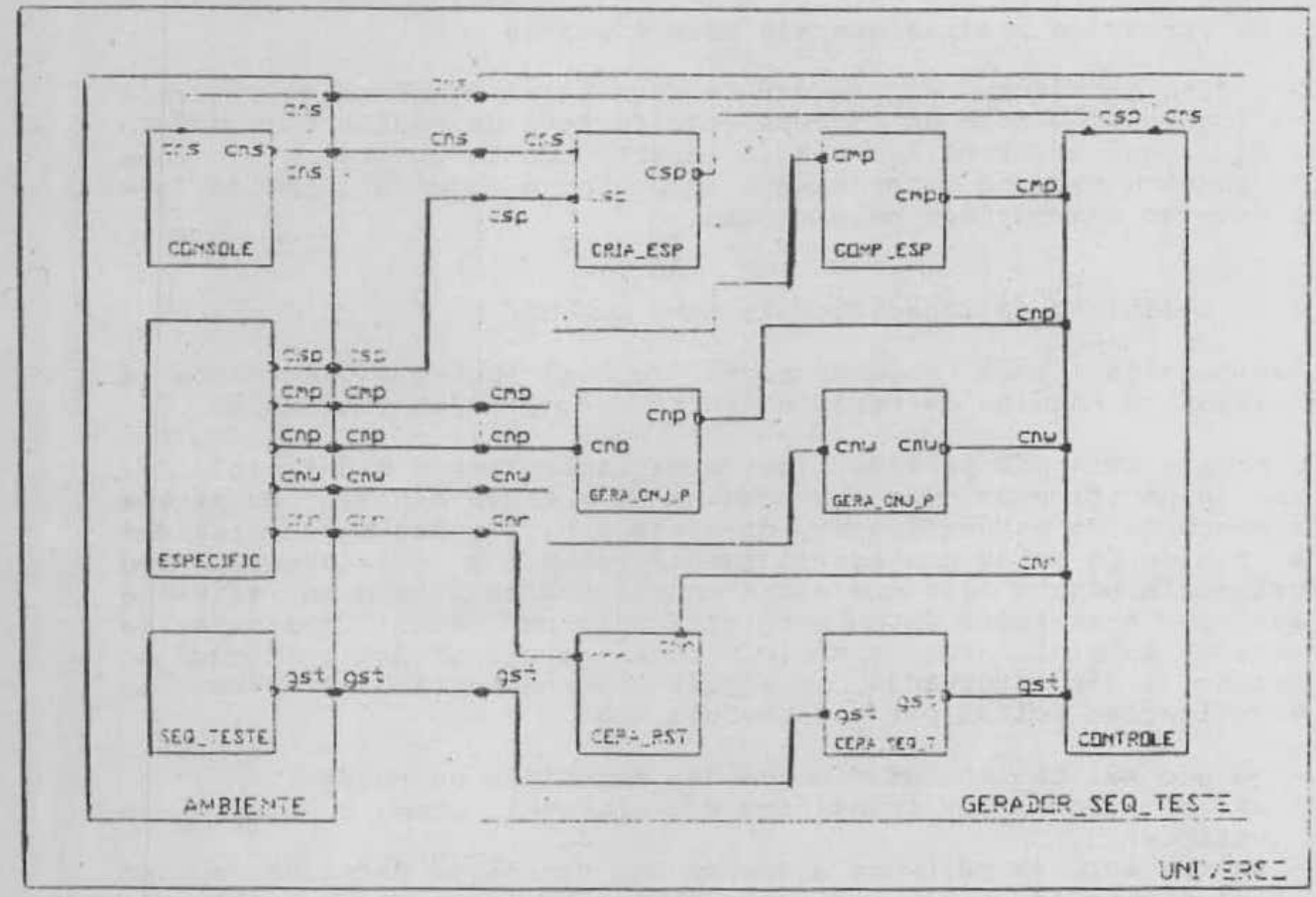


**Figura 6 - Modelo Estrutural do Gerador de Sequências de Teste**

A estrutura de dados utilizada para representar a máquina de estados finita baseia-se em uma matriz de adjacências. Cada posição desta matriz representa uma transição de estados, e possui uma lista de eventos de entrada/saída correspondentes a cada transição. Para grafos esparsos, a matriz de adjacências gera algum desperdício de memória, pois a matriz requer n x n posições de memória, onde n é o número de estados. Por outro lado, a utilização desta estrutura simplifica o armazenamento e as operações dos dados associados a ela.

## 4.4. Complementação da Especificação.

A especificação do protocolo de transporte classe 0, ilustrada na figura 1, não é definida completamente, ou seja, existem diversas entradas não especificadas para os estados da MEF. A geração de sequências de testes de um protocolo pode se basear em especificações incompletas, contanto que haja um método que suporte a geração destas sequências. A metodologia apresentada neste trabalho pode gerar sequências de teste a partir de especificações incompletas, desde que um conjunto W possa ser definido.

O módulo COMP_ESP tem como função principal complementar especificações definidas como máquinas de estados finitas. Este módulo constitui-se numa opção de projeto de geração das sequências de teste. A complementação da especificação facilita o teste de protocolos.

O módulo oferece duas alternativas para complementar as especificações:

1. Cada estado ignora qualquer entrada não especificada, permanecendo no mesmo estado.

2. Um "estado de erro" é criado. A construção deste estado segue os seguintes passos:

   a. um estado é adicionado ao modelo, denominado estado de erro;

   b. transições não especificadas para qualquer entrada e estado são direcionadas para o estado de erro;

   c. deve existir pelo menos uma transição que leva o protocolo para fora do estado de erro;

   d. o estado de erro deve ignorar todas as demais entradas e permanecer no estado de erro.

O módulo COMP_ESP possui interface com os módulos ESPECIFIC e CONTROLE.

## 4.5. Geração dos Caminhamentos Parciais (Conjunto P).

O módulo GERA_CNJ_P constrói o conjunto P em duas etapas distintas. Na primeira etapa, uma árvore de teste é criada, contendo todas as transições da MEF. [Kou,87] apresenta um refinamento deste algoritmo, onde os estados em cada nível da árvore são manipulados através de conjuntos. Isto diminui a complexidade do algoritmo. Neste trabalho, alguns modificações foram feitas na estrutura lógica deste algoritmo, a fim de torná-lo mais eficiente.

Na segunda etapa, extrai-se os caminhamentos parciais da árvore de teste, inclusive a sequência vazia, dando origem ao conjunto P. O algoritmo de construção deste conjunto é derivado do algoritmo Depth First Search (DFS) que, recursivamente, visita todos os nós da árvore de teste.

A estrutura de dados utilizada para representar a árvore de teste baseia-se em uma árvore Multiway [Wirth,76]. Cada nó da árvore armazena o valor de um estado e uma lista de entradas que representam as diversas arestas saindo daquele nó.

A complexidade para se gerar a árvore de teste é proporcional a (n.k), pois cada arco do grafo aparece apenas uma vez na árvore gerada. Como existem (n.k+1) caminhamentos parciais, pode-se concluir que a complexidade para se gerar o conjunto P é de $O(n\uparrow 2.k)$, onde n corresponde ao número de estados e k ao número de entradas.

## 4.6. Definição de um Conjunto W.

A geração do conjunto W é feita extraindo-se da especificação sequências de entradas que identifiquem os estados da máquina de forma única. O módulo GERA_CNJ_W tem como função gerar estas sequências de entrada. O algoritmo resultante do refinamento deste módulo baseia-se na técnica de "backtracking" [Wirth,76].

A estrutura de dados utilizada pelo módulo GERA_CNJ_W é uma matriz n x k, onde n representa o número de estados e k o número de entradas da MEF. Para cada estado n da MEF, o conjunto de entradas é pesquisado. Para cada entrada k deste conjunto, a saída do estado n é comparada com a saída de n-1 estados, a fim de se identificar uma entrada que caracterize o estado n. Portanto, a complexidade deste algoritmo é, no pior caso, de $O(n\uparrow 2.k)$. O resultado da aplicação do módulo GERA_CNJ_W no protocolo de transporte classe 0 é ilustrado na figura 7.

```
1 CR.[P1] 1          <-- Cnj W p/ Estado 1
2 NCONconf 3                 "          2
3 DR 1                       "          3
4 TCONresp 5                 "          4
5 DR 5                       "          5
```

Figura 7 - Sequências que compõem o Conjunto W

## 4.7. Identificação de "Resets" para a MEF.

A sequência W deve ser acompanhada de uma sequência de entradas que transfiram a implementação sob teste de um estado final, após a aplicação de W, para seu estado inicial. Estas sequências, denominadas "resets", podem ser geradas a partir da especificação do protocolo.

O módulo GERA_RST tem como função principal a geração de "resets" para a especificação definida como uma MEF. Este módulo possui interfaces com os módulos ESPECIFIC e CONTROLE.

O módulo GERA_RST baseia-se no algoritmo desenvolvido por R.W.Floyd [Aho et al.,83]. Este algoritmo, solução para o problema APSP (All Pairs Shortest Paths) em grafos dirigidos, utiliza uma matriz n x n para computar os comprimentos dos caminhos mais curtos. No caso de máquinas de estados finitas, onde as arestas são representadas por entradas, os caminhos entre dois vértices (estados) possuem valor constante.

Apesar do algoritmo gerar todos os caminhos mais curtos, apenas os caminhos de cada estado até o estado inicial são selecionados. A complexidade deste algoritmo é de $O(n^3)$.

As sequências geradas para o protocolo de transporte são ilustradas na figura 8.

```
2 TDISreq 1          <-- Resets p/ Estado 2
3 CC.[P6] 1                  "          3
4 TDISreq 1                  "          4
5 TDISreq 1                  "          5
```

Figura 8 - Sequências de Resets.

4.8. Geração das Sequências de Teste.

Para que o módulo GERA_SEQ_T possa gerar as sequências de teste, é condição básica que os módulos GERA_CNJ_W, GERA_CNJ_P e GERA_RST tenham sido executados. Um vez verificada a existência dos conjuntos gerados por estes módulos, cada sequência de teste é construída da seguinte forma:

a. Ler uma sequência P.
b. Identificar o último estado de P.
c. Concatenar uma sequência de W que identifique este último estado de P.
d. Identificar o último estado de W.
e. Concatenar uma sequência de "Reset" que leve a máquina do último estado de W até o estado inicial. Esta sequência só é agregada se o último estado de W for diferente do estado inicial.

Uma sequência de teste será o resultado da concatenação de uma sequência do conjunto P, de um sub-conjunto de W e de uma sequência de "reset". A figura 9 apresenta um trecho das sequências de teste geradas para o protocolo de transporte classe 0.

```
1 CR.[P1] 1
1 TDTreq 1 CR.[P1] 1
1 CC.[P8] 1 CR.[P1] 1
1 CC.[P6] 1 CR.[P1] 1
1 NCONconf 1 CR.[P1] 1
1 NRSTind 1 CR.[P1] 1
1 NDISind 1 CR.[P1] 1
1 TDISreq 1 CR.[P1] 1
1 TCONresp 1 CR.[P1] 1
1 CC 1 CR.[P1] 1
1 CR.[P1] 1 CR.[P1] 1
1 DT 1 CR.[P1] 1
1 ER 1 CR.[P1] 1
1 DR 1 CR.[P1] 1
1 TCONreq.[P0] 1 CR.[P1] 1
1 TCONreq.[P4] 2 NCONconf 3 CC.[P6] 1
1 TCONreq.[P4] 2 NDISind 1 CR.[P1] 1
1 TCONreq.[P4] 2 TDISreq 1 CR.[P1] 1
1 TCONreq.[P4] 2 TDTreq 2 NCONconf 3 CC.[P6] 1
1 TCONreq.[P4] 2 CC.[P8] 2 NCONconf 3 CC.[P6] 1
1 TCONreq.[P4] 2 CC.[P6] 2 NCONconf 3 CC.[P6] 1
```

Figura 9 - Algumas Sequências de Teste do Protocolo de Transporte Classe 0

Todos os módulos deste projeto foram e implementados na linguagem de programação Pascal, utilizando o compilador Turbo Pascal, versão 3.01. O programa fonte consiste de 1900 linhas, dando origem a 33 kbytes de código objeto.

## 5. Conclusões.

O projeto final das sequências de teste depende diretamente das características do testador e da forma como os testes serão conduzidos. Se o testador possuir uma arquitetura local, por exemplo, a introdução de predicados de habilitação nos eventos de entrada se torna viável, desde que exista um conjunto W. Esta viabilidade é decorrência das próprias características do testador local, onde o ambiente em que a implementação sob teste está inserida é totalmente controlado. Assim, as condições impostas pelos predicados de habilitação podem ser simuladas por este testador. Isto já não ocorre com um testador de arquitetura distribuída, uma vez que as condições do ambiente não estão inteiramente sob o controle do testador.

O problema da sincronização em arquiteturas de teste distribuídas também pode ser eliminado, dependendo do projeto do testador. Nos casos em que a iniciativa dos testes é sempre do testador ativo, as perdas de sincronismo durante a realização dos testes tendem a desaparecer. Um caso prático é apresentado em [Greco et al.,87], onde o respondedor possui um papel passivo durante a realização dos testes.

Ao se testar protocolos reais, o número de estados e entradas pode ser bem maior devido a considerações de implementação e variações dos parâmetros de interação das primitivas de entrada. Por causa disso, sempre existirá uma limitação na efetividade da realização dos testes de um protocolo de comunicação.

6 - Bibliografia.

[Aho et al.,83], - Aho,A., Hopcroft,J.E., Ullman,J.D., "Data Structures and Algorithms", Addison-Wesley, USA, 1983, 427 pg.

[Bochmann et al.,83] - Bochmann,G.v., Cerny,E., Maksud,M., Sarikaya,B., "Testing Transport Protocol Implementations", Proc. of CIPS'83, Ottawa, May 1983, pg. 123-129.

[Campos et al.,80] - Campos,I.M., Laender,A.H.F., Braga,J.L., "Estrutura e Transparência em Software: Encapsulamento através de Informação Redundante", Série de Monografias do Departamento de Ciência da Computação, n.T02/79, Belo Horizonte, 1980, 20 pg.

[Chow,78] - Chow,T.S., "Testing Software Design Modeled by Finite-State Machines", IEEE Transactions on Software Engineering, vol.SE-4, no.3, May 1978, pg. 178-187.

[Gill,62] - Gill,A., "Introduction to the Theory of Finite-State Machines", McGraw-Hill, New York, 1962, 207 pg.

[Giozza et al.,86b] - Giozza,W.F., Moura,J.A.B., Sauve,J.P., Araújo, J.F.M., "Redes Locais de Computadores - Protocolos de Alto Nível", McGraw-Hill, São Paulo, 1986, 446 pg.

[Greco et al.,87] - Greco,C.E., Bion,E.C.B., Gerber,G.W., Filho,L.A., "Teste de Implementações de Protocolos de Comunicação", XIV Seminário Integrado de Software e Hardware - VII Congresso SBC Salvador, Julho 1987, pg. 602-611.

[Kohavi,78] - Kohavi,Z., "Switching and Finite Automata Theory", Mc Graw-Hill, New York, 1978, 658 pg.

[Kou,87] - Kou,T., "Conformance Testing of OSI Protocols: The Class 0 Transport Protocol as an Example", Tese de Mestrado, University of British Columbia, Canada, August 87, 79 pg.

[Linn & Nightingale,83a] - Linn,R.J., Nightingale,J.S., "Some Experience with Testing Tools for OSI Protocol Implementations", Report no. ICST/SNA 83-1, June 1983, 11 pg.

[Rafiq,86] - Rafiq,O., "A Good Approach for Testing Protocol Implementations", Proc. of the International Seminar on Computer Networking - Japan, 1986, pg. 305-314.

[Rayner,82] - Rayner,D., "A System for Testing Protocol Implementations", Proc. IFIP WG 6.1 2nd International Workshop on Protocols, 1982, pg. 383-395.

[Sarikaya,87] - Sarikaya,B., "Protocol Testing: Architectures and Test Sequences", Concordia University, Montreal, September 1987, 35 pg.

[Wirth,76] - Wirth,N., "Algorithms + Data Structures = Programs", Prentice-Hall Inc., New Jersey, 1976, 365 pg.