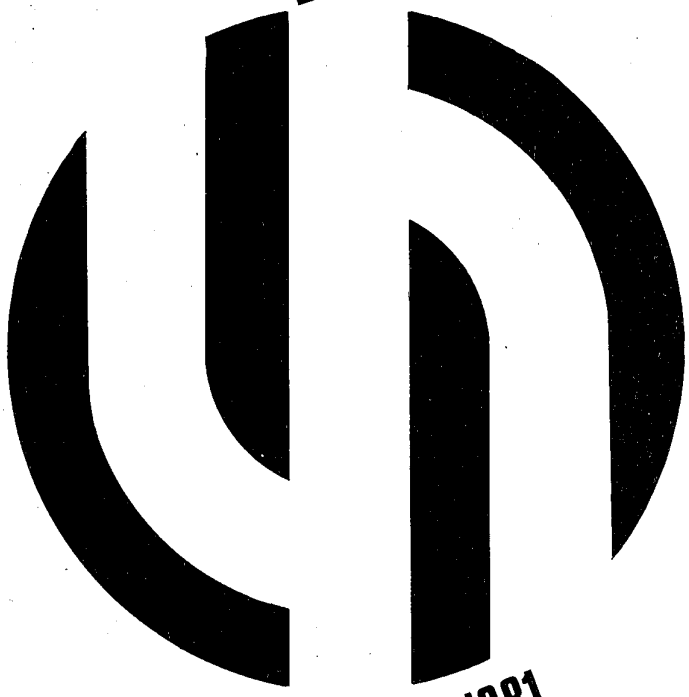
# anais do 8º seminário integrado de SOFTWARE E HARDWARE



## 27 A 31 de julho de 1981
## ufsc · florianópolis



Sociedade Brasileira de Computação

Conselho Nacional de Desenvolvimento Científico e Tecnológico

SEI / Secretaria Especial de Informática

CAPES / Coordenação de Aperfeiçoamento de Pessoal do Ensino Superior

UFSC / Universidade Federal de Santa Catarina

Governo do Estado de Santa Catarina

ANAIS DO

VIIIº SEMISH

SEMINARIO INTEGRADO DE SOFTWARE E HARDWARE

Iº CONGRESSO DA

SOCIEDADE BRASILEIRA DE COMPUTAÇÃO

FLORIANÓPOLIS, SANTA CATARINA

27 A 31 DE JULHO DE 1981

# PROVA DE PROGRAMA COM TRATAMENTO DE EXCEÇÕES EM AMBIENTE INDUSTRIAL

Arndt von Staa

Departamento de Informática
Pontifícia Universidade Católica
Rio de Janeiro

SUMÁRIO

Prova de correção de programas é geralmente considerada antieconômica em ambiente industrial. Muitas vezes, no entanto, nós nos defrontamos com programas "resistentes" a testes. Ou seja, os métodos de teste tradicionais são inadequados para alcançar um grau de confiança satisfatório. Ora, tal não é admissível em um software de alta precisão.

Uma das causas da resistência ao teste é a imprevisibilidade da sequência de execução. Esta imprevisibilidade é natural em programas interativos onde o operador tem ao seu dispor funções que permitam, a qualquer instante, mudar a sequência de processamento.

Este artigo apresenta um modelo para a prova da correção de programas construídos de forma modular e onde é natural o tratamento de exceções.

Palavras Chave

Entrada de dados
Máquina de estados
Modularidade
Prova informal da correção
Tratamento de exceções.

## 1. INTRODUÇÃO

A prova de correção de programas tem sido, frequentemente , rotulada como um exercício acadêmico de baixa aplicabilidade prática. Tal rótulo pode advir de uma falta de entendimento dos fundamentos teóricos e, consequentemente, da dificuldade virtualmente intransponível encontrada ao efetuar-se a prova. Pode advir dos exemplos usados, muitas vezes pouco representativos por tratarem de programas "obviamente corretos". Pode advir, finalmente, do excesso de zelo por parte do "provador", levando a provas muito detalhadas e, portanto, demoradas, onde uma prova informal seria suficiente para alcançar o mesmo resultado.

Tendo em mente este pano de fundo, torna-se interessante relatar todos os esforços, bem sucedidos ou não, em que foi efetuada uma prova de correção de programa em ambiente industrial.

A experiência aqui relatada foi bem sucedida em dois aspectos. Primeiro, o objetivo direto, a prova informal de um programa, levando a um programa produto de elevado grau de confiabilidade. Segundo, um objetivo secundário, mostrar um exemplo de uma colaboração empresa/universidade bem sucedida.

## 2. DESCRIÇÃO DO PROBLEMA

Nesta seção descreveremos, em linhas gerais, o sistema de transcrição de dados do qual faz parte o programa a ser provado.

Sistemas de transcrição de dados dão suporte mecanizado à transcrição de documentos (lotes fonte) legíveis por pessoas, para registros legíveis por meios mecânicos (lotes transcritos). A acuidade e a fidelidade[1] dos resultados, os lotes transcritos , deve ser a maior possível, uma vez que diversos outros sistemas utilizarão direta ou indiretamente estes dados. Assim, diversos controles de validade são impostos a estes dados. Estes controles são utilizados tanto para elevar o grau de acuidade do processo manual de preparação dos documentos, quanto para assegurar uma elevada fidelidade ao processo de transcrição, também efetuado manualmente. Devido à existência destes controles, erros de transcrição

(1) a acuidade mede a precisão e correção do processo de coleta de dados e, também, a correção do preenchimento de formulários. A fidelidade mede ao grau de igualdade entre os dados fornecidos e os transcritos.

e/ou de preparação de dados são descobertos durante o próprio processo de transcrição tão logo se tornem aparentes. O digitador poderá, então, corrigir os erros de transcrição, ou anotar no próprio formulário erros de preenchimento. Após o acerto dos formulários em erro, o lote transcrito será editado, sendo os controles de validade efetuados de novo. A transcrição de dados assim efetuada, acelera substancialmente o processo de preparação de dados de elevada qualidade, quando comparada com métodos tradicionais.

Em um ambiente interativo de transcrição de dados o digitador tem ao seu dispor diversos comandos que habilitam a alterar o fluxo natural de transcrição e edição. Tal é necessário para permitir a edição (correção) de dados incorretamente transcritos ou preenchidos nos documentos de entrada.

A validação desejada é expressa através de um programa de crítica. Para cada serviço de transcrição existirá um programa de crítica próprio. Este programa define tanto os dados que serão transcritos, quanto a validação destes dados.

Devido à possibilidade de alterar o fluxo normal de transcrição e a possível ocorrência de diversas sessões de edição de um lote transcrito, é impossível, no caso geral, saber-se, a priori, a sequência de ações utilizada pelo digitador para alcançar determinado ponto no processo de transcrição. Entretanto, o nível de qualidade dos dados deve ser insensível a esta sequência de ações. Ou seja, dada uma definição de qualidade através de um adequado programa de crítica, a qualidade dos dados transcritos por intermédio deste programa deverá depender somente dos dados em si, não devendo depender do processo utilizado para a sua transcrição. Caberá, assim, somente ao programa de crítica efetuar o controle da fidelidade de transcrição e da acuidade da preparação dos dados, cabendo nenhuma responsabilidade ao digitador ou a outras pessoas que porventura tenham participado do processo de transcrição/edição dos dados. Consequentemente, para o programador, se o lote transcrito é válido segundo o que foi especificado, isto será verdadeiro, independentemente da sequência de ações utilizada pelo digitador para transcrever este lote.

O sub-sistema de controle da transcrição deverá atender, aos seguintes requisitos específicos:

1. resultado:

- os dados transcritos e o correspondente diagnóstico deverão sempre estar de acordo com o programa de crítica definido pelo usuário.
- caso o dado seja válido, o correspondente diagnóstico elementar deverá estar nulo.
- para um mesmo conjunto de dados transcritos e um mesmo programa de crítica, deverá ser produzido o mesmo diagnóstico independente da sequência de ações utilizada pelo digitador para produzir tal conjunto de dados.
- o conteúdo da tela do terminal de transcrição deverá ser uma cópia fiel da porção do lote transcrito em exibição no terminal.

2. utilizabilidade

- todos os campos são editáveis, devendo ser assegurado que o diagnóstico e a tela corresponda sempre aos valores mais recentes dos dados conforme registrados no lote transcrito.
- caberá somente ao programador a responsabilidade de definir critérios de validade bem como as necessárias redundâncias internas dos dados, que assegurem um nível de qualidade adequado.
- o programador poderá definir ações a serem tomadas em caso de erro. Estas ações poderão ser: i) registrar um erro no diagnóstico e prosseguir; ii) exigir a redigitação ou iii) rejeitar o campo, registrando o fato no diagnóstico.
- no caso de erros que exijam a redigitação, a interrupção do ciclo de redigitação ocorrerá por: i) digitação de dado válido; ii) término do ciclo através de condição pré-estabelecida pelo programador (por exemplo, número de redigitações maior do que n); ou iii) rejeição de campo efetuada por ação do digitador. Neste último caso será registrada uma mensagem criada pelo digitador no diagnóstico correspondente.

## 3. TRATAMENTO DE EXECEÇÕES EM AMBIENTES MODULARES

Nesta seção será abordado, de forma resumida, o método utilizado para provar a correção de programas modulares capazes de pro

cessar execeções detectadas e sinalizadas entre e/ou intra módulos. O método utilizado é semelhante ao descrito em [1].

Módulos são representados por máquinas de estado (automatos) . As transformações de dados (processamento) ocorrem nos estados destas máquinas. Transita-se de um estado para outro em conformidade com as condições resultantes destas transformações de dados. Estas condições poderão corresponder à detecção de exceções ou, então, ao término normal do processamento.

Para podermos provar a correção de programas, necessitamos de assertivas. Estas são expressões lógicas (por exemplo relações) envolvendo as variáveis do programa conhecidas ao módulo e, possivelmente, variáveis auxiliares do próprio modelo de prova utilizado . Uma assertiva terá sido satisfeita, se e somente se a sua avalia - ção resultar verdadeira. Cade observar que esta avaliação é feita de forma simbólica durante o processo de prova.

Assertivas poderão ser agrupadas em conjuntos de zero ou mais assertivas. Um conjunto de assertivas terá sido satisfeito se e somente se todas as assertivas deste conjunto tiverem sido satisfeitas.

Cada estado possui zero ou mais pontos de entrada (zero se o estado é a origem do processamento ). A cada um dos pontos de entrada de um estado é associado um conjunto de assertivas. A máquina estará operando corretamente somente se todas as transições efetuadas assegurarem a satisfação do conjunto de assertivas de entrada correspondente.

Cada estado possui ainda zero ou mais transições de saída (zero se estado final). A cada transição de saída de um estado é associado um conjunto de assertivas de saída.

Do ponto de vista conceitual, módulos e estados representam o mesmo conceito. Assim, qualquer estado X poderá ser detalhado através de uma máquina de estados com os mesmos conjuntos de asserti - vas de entrada e de saída do estado X, e onde a transformação efetuada por X é descrita por esta máquina de estados detalhe. De forma semelhante, qualquer máquina de estados Y poderá ser fundida em um único estado efetuando a mesma transformação que Y, e possuindo os mesmos conjuntos de assertivas de entrada e saída que a máquina Y.

O tratamento externo de exceções é modelado representando-se o módulo que detecta a exceção como um estado em uma máquina envolvente. Este estado, ativado pela exceção, é o tratador externo da exceção.

A prova de correção consiste em demonstrar, para cada estado e para cada conjunto de assertivas de entrada, que a transformação de dados efetuada no estado assegura a satisfação de um único conjunto de assertivas de saída deste estado. Além disso, devemos demonstrar, para cada transição, que a satisfação do conjunto de assertivas de saída definido na origem desta transição, assegura a satisfação do conjunto de assertivas de entrada associado ao ponto de entrada atingido por esta transição.

A exigência de que no máximo um dos conjuntos de assertivas de saída seja satisfeito, assegura o determinismo na execução da máquina. A exigência de que no mínimo um dos conjuntos de assertivas de saída seja satisfeito, assegura a continuidade da execução desta máquina. Ambas estas exigências valem para todos os estados e todas as transições da máquina em questão, exceto o estado final.

É óbvio que poderá ser difícil encontrar os diversos conjuntos de assertivas de forma a descrever completamente o programa que queremos provar. Sem assegurar esta completeza, o resultado da prova, por mais formal que seja, poderá estar errado sem que o saibamos. Além disto a prova informal pode omitir acidentalmente pontos relevantes.

Uma das vantagens de conjugar-se a prova informal da correção com teste sistemático, é o grau de confiança que o programador adquire com relação ao seu programa. Uma outra vantagem é a possibilidade de eliminar-se erros antes mesmo de redigir-se a primeira linha de código. Isto é o caso quando se utiliza o modelo de prova como modelo de especificação do programa. Estatísticas relativas a provas de correção de programas revelam que 80% dos erros podem ser eliminados, se o modelo de prova é utilizado como modelo de implementação.

## 5. EXEMPLO

Nesta seção será apresentado um exemplo de prova informal de parte do sub-sistema de controle de transcrição. Este sub-sistema

é particionado nos seguintes módulos hierárquicos:

- transcreve lote (conjunto de documentos)
- transcreve documento(conjunto de campos)
- transcreve campo.

Por razões de espaço iremos nos limitar no presente trabalho a:

- apresentar somente o módulo transcreve campo;
- apresentar somente assertivas relacionando campos transcritos (campo, memória), campos na tela (campo.tela)e diagnóstico relativo ao valor do campo (campo.diagnóstico).
- apresentar a máquina de estados sem preocupação com o seu desempenho.

Cabe salientar aqui que na prova real adotada, todas as considerações relativas a desempenho e a variáveis de interface foram efetivamente incluídas, não constituindo, portanto, uma limitação do modelo.

Na figura 1 apresentamos o diagrama da máquina de estados do módulo "transcreve campo" a ser provado. Neste diagrama cabe salientar que a transformação interna ao estado 2 é parcialmente fornecida pelo usuário através do programa de crítica.

A seguir descreveremos os diversos conjuntos de assertivas utilizados na prova. Nestes conjuntos utilizaremos as seguintes convenções:

- os conjuntos de assertivas de entrada relacionarão somente as variáveis a serem utilizadas, pela transformação de dados efetuada no estado correspondente. Assim, se determinada variável não é relacionada no conjunto de assertivas , de entrada ela poderá possuir qualquer valor.
- os conjuntos de assertivas de saída, relacionarão variáveis alteradas pela transformação interna ao estado.

É seguinte a descrição da máquina:

Ponto de entrada: "transcreve campo"

. função: lê e valida campo que ainda não havia sido transcrito.

. assertivas de entrada:

. id.campo - identifica o campo a ser transcrito

. modo - igual a "transcrição"

. campo.memória, campo.tela e campo.diagnóstico estão todos nulos.

Ponto de entrada: "edita campo"

. função: lê e valida campo já transcrito anteriormente.

. assertivas de entrada:

. id.campo - identifica o campo a editar

. modo - igual a "edição"

. campo.memória e campo.tela - iguais

. campo.diagnóstico - contém o diagnóstico gerado de acordo com o programa de crítica (ou ação de digitador: rejeita campo) e o valor contido em campo.memória. Se o campo é válido, o diagnóstico é nulo.

Ponto de saída: "retorna valor" (Estado 4)

. condição: o retorno ocorre através deste ponto se e somente se foi digitado um valor ou foi acionado avança campo em modo "edição".

. assertiva de saída:

. campo.memória e campo.tela - iguais

. campo.diagnóstico - em correspondência com o valor de campo.memória.

. campo.id e modo - inalterados

. ação - igual a "valor"

Ponto de saída: "retorna ação" (Estado 5)

. condição: o retorno ocorre através deste ponto de retorno se e somente se o digitador acionou uma tecla de função.

. assertiva de saída

. campo.memória, campo.tela, campo.diagnóstico

. id. campo e modo - inalterados

. ação - igual ao código da tecla de função acionada (este código é diferente de "valor").

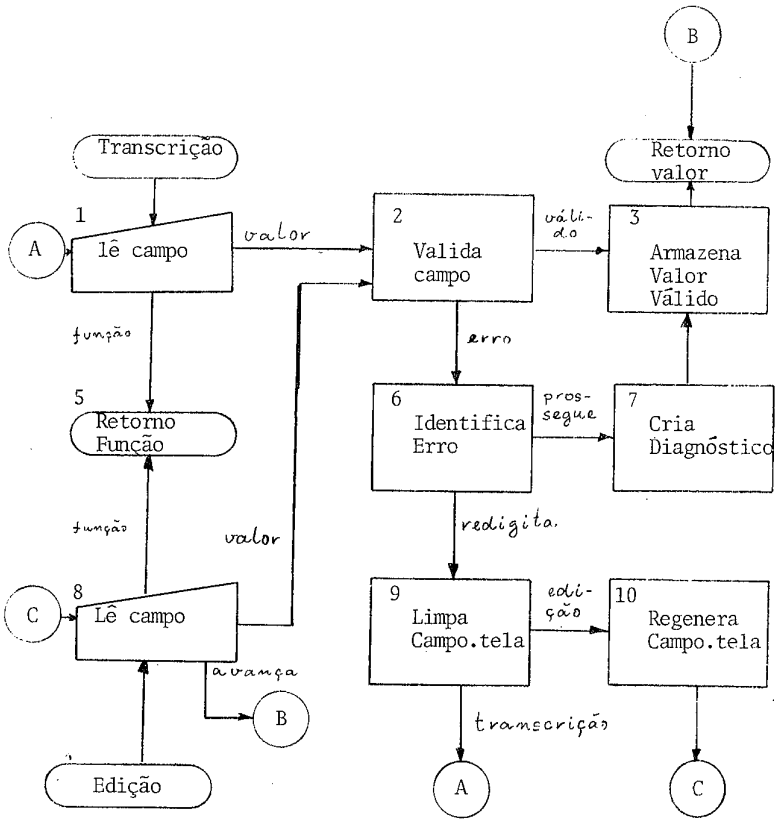



Figura 1: DIAGRAMA DE ESTADOS

Estados de transformação:

Estado 1: "lê campo"

. função: espera por dado a ser fornecido pelo digitador, quando em modo "transcrição". Este dado pode ser: "valor" ou "tecla de função".

. assertiva de entrada

. campo.memória, campo.tela e campo.diagnóstico - todos nulos

. modo - igual a "transcrição"

. conjuntos de assertivas de saída

. transita para estado 2 se campo.valor for "valor"

. campo.valor - igual ao valor digitado

. ação - igual a "valor"

. transita para estado 5 se foi acionada tecla de fun - ção

. campo.valor - igual ao código da tecla acionada

. ação - igual a "tecla de função"

Estado 2: "valida campo"

. função: efetua a validação do valor digitado, segundo o programa de crítica fornecido

. assertiva de entrada

. campo.valor - igual ao valor digitado

. campo.tela - igual ao valor digitado

. campo.memória, campo.diagnóstico - inalterados

. assertivas de saída

. transita para 3 se válido

. validade - igual a "válido"

. transita para 6 se não válido

. validade - igual ao código do erro detectado

Estado 3: "armazene valor"

. função: óbvia.

. assertiva de entrada

. validade - igual a "válido", ou, então, a código de erro que não requeira redigitação

. campo.valor - igual a valor digitado

. campo.diagnóstico - em correspondência com campo.valor

. campo.tela - igual a campo.valor

. assertiva de saída

. transita para o estado 4.

. campo.memória - igual a campo.valor

Estado 6: "identifique erro"

. função: determinar se o erro exige ou não a redigitação.

. assertiva de entrada

. validade - igual ao código do erro detectado

. assertiva de saída

. transita para estado 7 se erro não requer redigitação

. transita para estado 9 se erro requer redigitação

Estado 7: "gere diagnóstico"

. função: registrar o erro no diagnóstico

. assertiva de entrada

. validade - igual ao código do erro detectado

. assertiva de saída

. transita para estado 3

. campo.diagnósitco - igual a código do erro detectado

Estado 8: "lê campo"

. função: espera por dado a ser fornecido pelo digitador quando em modo "edição". O valor poderá ser "valor", "tecla de função", ou ação "avança campo" (valor nulo).

. assertiva de entrada

. ver ponto de entrada "edita campo"

. assertiva de saída

. transita para estado 4 se valor é nulo (avança campo)

. campo.tela, campo.memória e campo.diagnóstico iguais ao valor fornecido ao entrar no módulo

. campo.valor - igual a nulo

. transita para estado 2 se valor não nulo

. campo.tela - igual a valor digitado

. campo.valor - igual a valor digitado

. transita para estado 5 se foi acionada tecla de função

. ação - igual a "tecla de função"

. campo.valor - igual ao código da tecla acionada

Estado 9: "limpe campo.tela"

. função: óbvia.

. assertiva de entrada

. irrestrito

. assertiva de saída

. transita para estado 1 se modo igual a "transcrição"

. transita para estado 10 se modo igual a "edição"

. campo.tela - igual a nulo

. demais variáveis satisfazem assertivas de entrada "edição campo"

Estado 10: "regenere campo.tela"

. função : óbvia

. assertiva de entrada

. campo.tela - igual a nulo

. assertiva de saída

. transita para estado 6

. campo.tela - igual a campo.memória

. demais variáveis - satisfazem assertiva de entrada "edita campo"

Examinando a descrição acima, deve ser fácil ao leitor certificar-se que o programa faz o que se deseja, exceto no que tange às transições de saída do estado 8. Vejamos pois:

8 -> 5 neste caso não há anormalidade, uma vez que as variáveis do campo não foram alteradas.

8 -> 2 neste caso surge o problema que campo.diagnóstico deve ser nulo ao entrar no estado 2 e, no entanto, não o é ao sair do estado 8. Se anularmos campo.diagnóstico, o retorno ao estado 6 é inválido, pois campo.diagnóstico não estará mais necessariamente em acordo com campo.memória.

Claramente isto é um erro, pois o resultado do módulo "transcreve campo" não estará correto se o digitador agora acionar avança campo. Uma solução seria salvar o valor de entrada de campo.diagnóstico e restaurar esta variável para o valor salvo no caso de uma ação "avança campo". Fazer estas alterações no diagrama e na descrição é fácil e, portanto, não será feito aqui.

8 -> 4 também resultaria em erro, caso campo.diagnóstico fosse simplesmente anulado. No caso de ser salvo, antes de retornar (estado 4), o valor original de campo.diagnóstico deverá ser restaurado. Isto implica na adição de um estado à máquina.

## 6. CONCLUSÃO

O presente método foi utilizado para demonstrar informalmente a correção do sub-sistema de controle de transcrição DELPH, um programa MUMPS de 3k bytes de extensão. Foram seguintes as obser-

vações feitas durante o processo de certificação e verificação.

i. pode-se descobrir e circundar uma inadequação da especificação do terminal utilizado. Esta inadequação causava erros intermitentes e dependentes da carga de CPU.

ii. a prova foi efetuada após um já considerável esforço de teste. Apesar disto, diversos erros foram detectados e removidos durante a prova.

iii. o esforço para definir as diversas máquinas de estado ( 5 ao todo), os diversos estados (cerca de 50) e os diversos conjuntos de assertivas (cerca de 80) levou cerca de 3 dias úteis. A prova em si foi trivial devido ao cuidado com que os conjuntos de assertivas foram gerados.

iv . após a prova foram conduzidos testes sistemáticos, não logrando encontrar erros (esforço: um dia útil).

v . o programa encontra-se em produção desde dezembro de 1980, não tendo sido reportado erro até agora.

Além do resultado imediato, a certificação de um programa industrial, a criação do modelo de prova necessário, motivou uma pesquisa no Departamento de Informática. Esta teve o intuito de demonstrar que o modelo de prova utilizado é geral e válido. Com este resultado[1], passa-se a ter certeza de que o modelo permite provar a correção do tratamento de exceções em programas modulares, problema até há pouco em aberto[2]. O presente trabalho evidencia, portanto, uma interação universidade/indústria cujo resultado foi benefício para ambas as partes.

Agradecimento

Desejo aqui agradecer ao Sr. Francisco Eduardo Rego Ramalho, diretor da Medidata S.A., por ter dado a oportunidade para a interação da qual resultou este trabalho.

Referências Bibliográficas

[1] - Azeredo, P.A.
Tratamento de exceções em ambientes modulares: Tese de Doutorado, Departamento de Informática, PUC/RJ, 1980.

[2] - Berry, D.M.; Kemmerer, R.A.; Staa,A.v.; Yemmini, S.
'Towards modular verifiable exception handling'; em Journal of Computer Languages,vol 5;Pergamon Press; 1980;pags. 77-101.