Sistemas Tolerantes a Falhas

Estruturando a Redundância em Sistemas TFs

Prof. Jó Ueyama

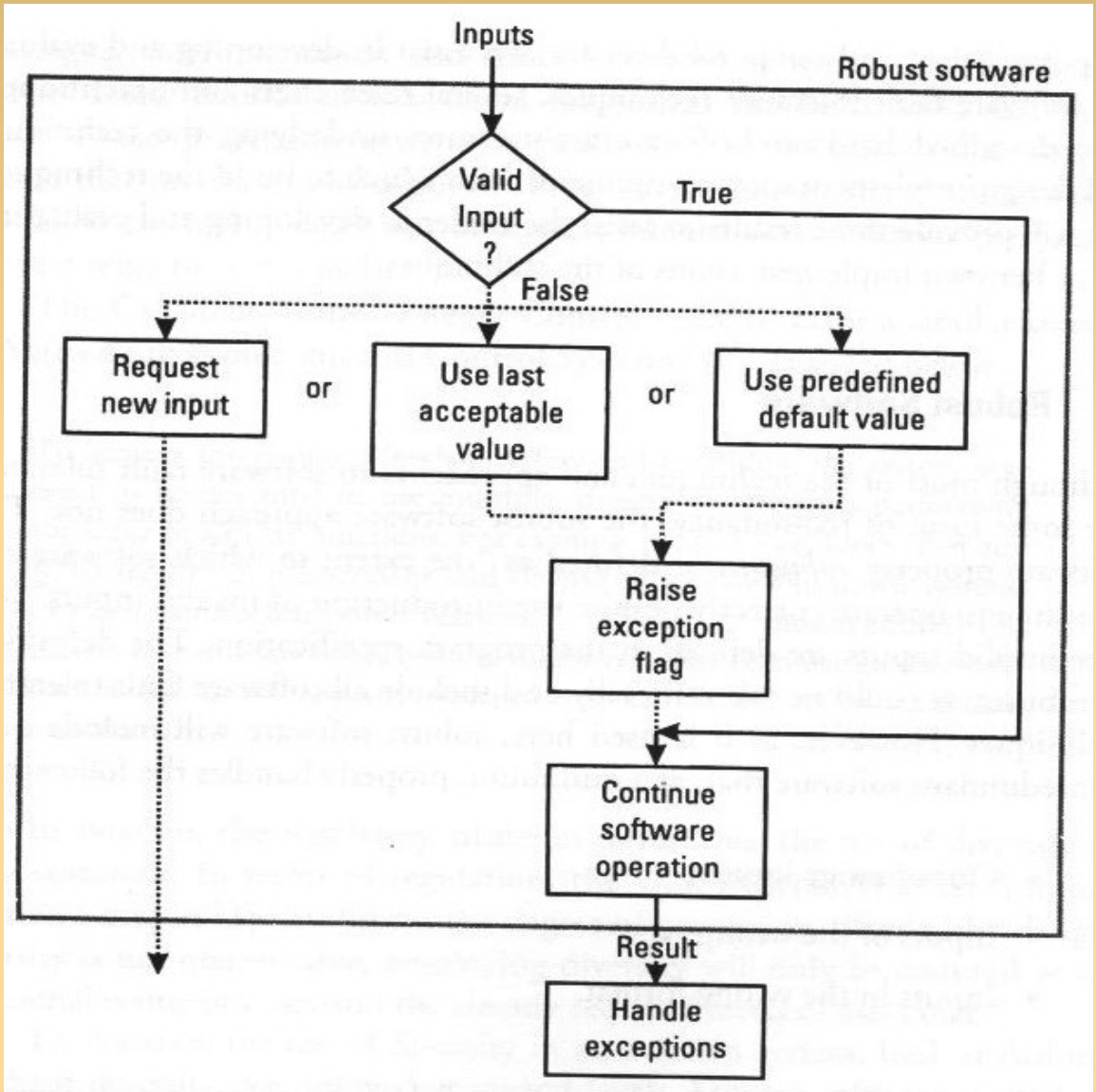
Redundância de Software

- Redundância e diversidade (diversity)
  - Qual a diferença?
  - Algum tipo de diversidade deve acompanhar a redundância
- A *US Nuclear Regulatory Agency* define normas de diversidade em sistemas redundantes
- Diversity é utilizado em sistemas de controle de voo em aviões Airbus A320, A330 e A340
- O controle da aeronave A320 utiliza dois tipos de computadores com arquiteturas e processadores distintos provenientes de fabricantes diferentes

Softwares Robustos

- Antes um breve comentário sobre os softwares robustos
- Sem redundância
- Pode operar mesmo que haja *inputs* inválidos
  - *Out of range inputs*
  - *Inputs of the wrong type*
  - *Inputs in the wrong format*
- Vantagem: detectar falhas 'cedo'
- Desvantagem: falhas só de *input*

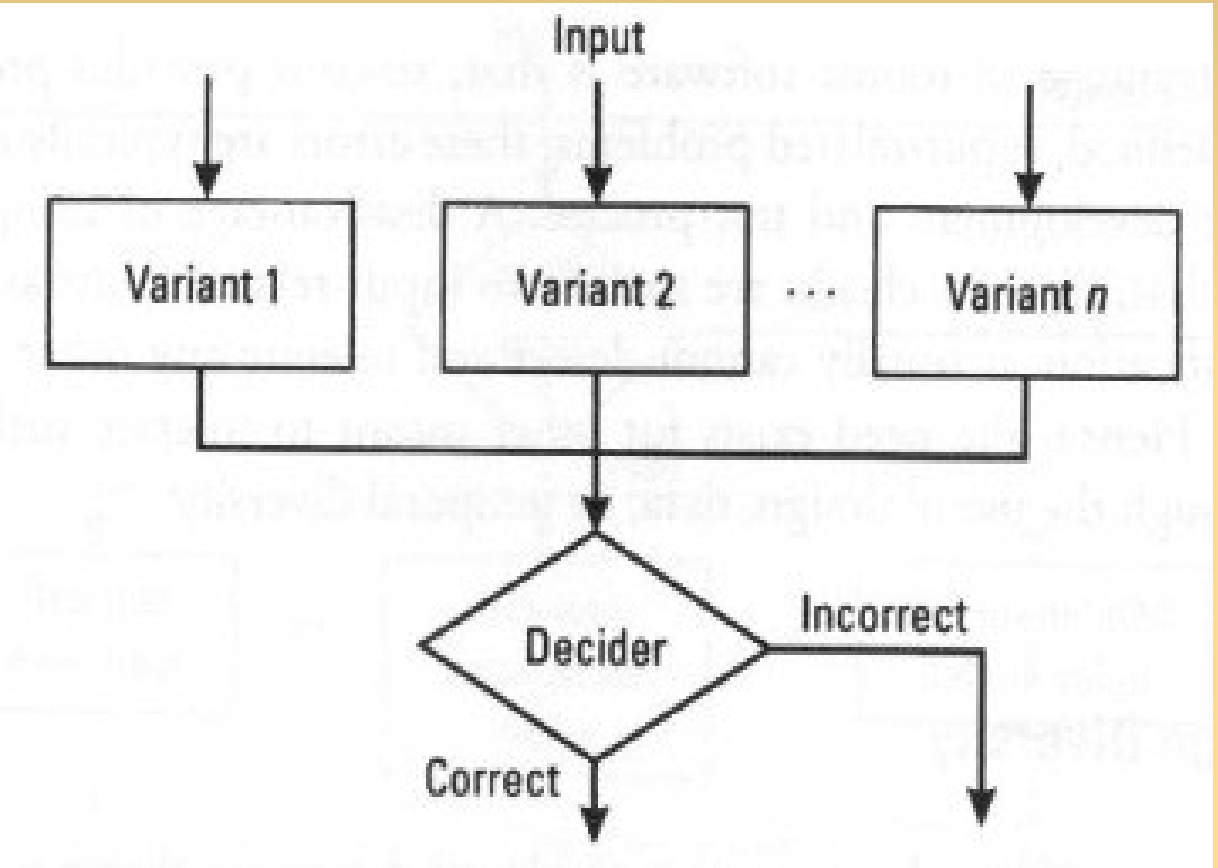
Operação de um Software Robusto



Projeto de Diversidade

- Serviços idênticos providos através de projeto e implementação 'separados'
- Chamados de módulos, versões ou variantes
- Minimizar erros 'idênticos'
- Garantir que pelo menos uma variante possa 'funcionar' sempre
- O 'juiz' decide qual das múltiplas saídas será encaminhada para a prx parte do sistema
  - Baseda na saída considerada aceitável ou correta

Projeto Básico de Diversidade



Projeto de Diversidade

- Um simples 'juiz' pode ser adotado qd existem uma grande independência entre as variantes
- Grande independência é dificilmente alcançada
- Na prática tem sido utilizado principalmente em sistemas embarcados críticos
- É um mecanismo muito caro
- Em torno de 1,7 a 0,85 vezes mais caro
- Por que não o dobro?
  - Porque nem tudo é realizado de forma separada, como a definição de requisitos

Algumas conclusões obtidas

- A maior parte dos erros causam de erros de especificações (e.g. incompleto ou ambíguo)
- Linguagens de programação mais 'rígidas' influenciam em falhas em NVP (e.g. Assembly)
- O uso do N-version tem ajudado a reduzir erros comuns
- Falhas são mais causadas pelos erros de projetos
- Existem casos de múltiplas especificações, mas a definição de requisitos é única

Níveis de Diversidade

- Decisão chave: que componentes serão 'diversificados'?
- Tradeoff entre pequenos e grandes componentes
- Pequenos são mais fáceis de administrar, menos complexos
  - inclusive da criação dos 'juízes'
- Grandes componentes são mais favoráveis para uma diversidade eficiente

Níveis de Diversidade

- Diversidade pode ser aplicada em diversos ambientes (hardware e software – aplic e básico)
- Quando adotado em várias camadas, então é chamada de *multilayer diversity*
- Em hardwares pode tolerar falhas em componentes de hardware, pois são fabricados separadamente
- Porém, mais aplicado no nível de softwares
  - Para assegurar continuidades de uma variante pelo menos
- Existem diversidades de hardware e software
  - *Dual or triple displays*

Níveis de Diversidade

- Existem vários exemplos de sistemas de multiversão de hardware e software
  - computadores de controle de voo em Boeing 737-300, Airbus A320, A330 e A340
- Desvantagem do *multilayer diversity*
  - Custo e performance
- Pode tornar-se proibitivo em sistemas de tempo-real (i.e., performance)
- Uso de *diversidade sistemática* para diminuir o custo do desenvolvimento com diversidade

Níveis de Diversidade

- Exemplos de diversidade sistemática
  - Uso de diferentes registradores (de uma mesma arquitetura de processador) por cada variante distinta
  - Transformação de expressões matemáticas
  - Uso de otimização de código, geradores de código e compiladores diferentes
  - Uso de bibliotecas diferentes
  - Implementar variantes com estruturas de programações diferentes
  - Uso de memória RAM de maneira diferenciada

Diversidade de Dados

- Limitações de outros tipos de diversidades levaram a diversidade de dados
- É um complemento de outros tipos
- Utiliza mecanismos de decisão em alguns pontos de execução do sistema
- Data re-experession
  - Obter variantes de dados de entrada logicamente equivalentes
- Diversidade de dados usa os DRA (*Data Re-expression Algorithms*)
  - Fornece variantes de dados de entrada

Diversidade de Dados e DRAs

- Técnicas usadas com a diversidade de dados
  - N-copy programming (NCP)
  - Retry block (RtB)
- A performance da diversidade de dados depende (*heavily*) nos DRAs
- DRAs são bem *application dependent*
  - Requer uma análise do tipo e magnitude da re-expressão apropriada para cada dado 'candidato'
- Uma variedade de DRAs já existe para diversos tipos de aplicações
  - Normalmente, eles são simples (evitar falhas neles)

DRAs

- DRAs são usados em sistemas de tempo-real - sensores que podem ler dados corrompidos
  - Tais dados poderão sofrer *re-expression* para tolerar falhas
  - Pequenas modificação nos dados não afetam o funcionamento do sistema
- Nem todas as aplicações podem usar a diversidade de dados
  - Aquelas aplicações que não possuem um DRA eficiente
  - E.g. aqueles que não manipulam muito com dados numéricos

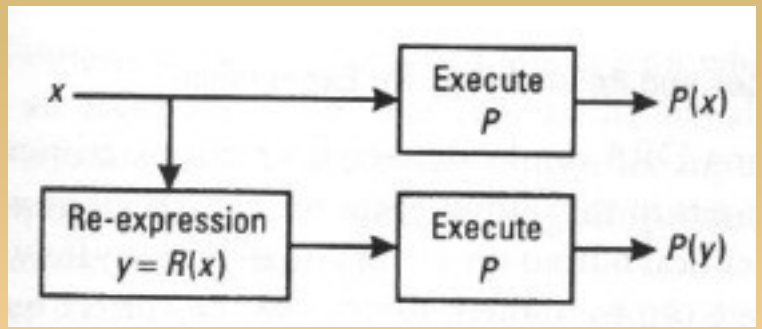
DRAs

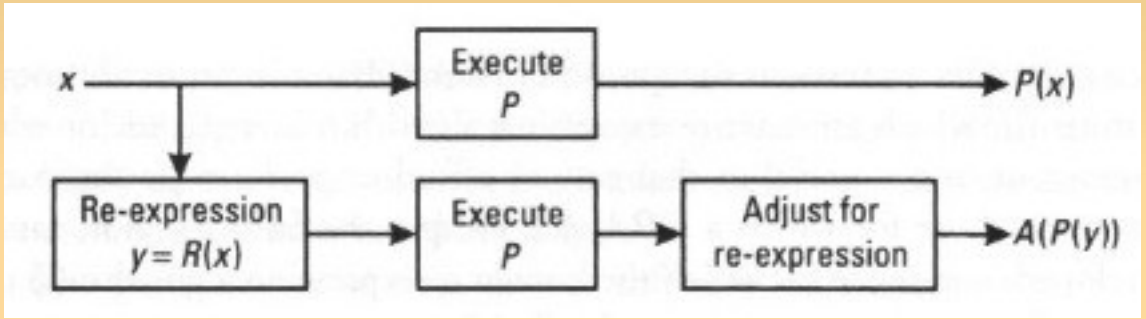
- Contexto em que os DRAs tornam-se ineficientes:
  - Um alto uso de valores estritamente inteiros
  - Aproximações não permitidas
  - Ajustes *postexecutions* não permitidas
- Ajustes pós-execução são realizadas fora das regiões de falhas
- Um *domínio de falha* é o conjunto de pontos de entrada de dados que ocasionam falhas
- A região da falha é a *geometria* (localização dentro do código) do *domínio da falha*

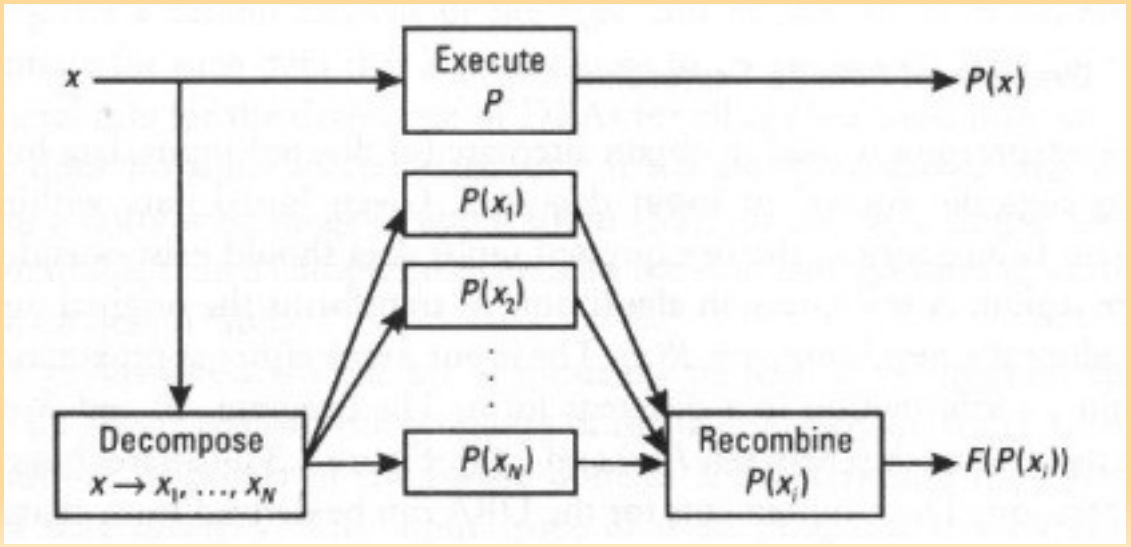
Overview da Re-expressão de Dados

- A re-expressão é:
  - Obter dados de entrada alternativos que sejam logicamente equivalentes
- Um algoritmo *R* toma *x* como dado de entrada e deriva *y*
  - *y* pode ser uma aproximação de *x* ou pode expressar *x* de uma maneira diferente
- Os requisitos de um DRA podem ser definidos pelas características da saída do sistema
- Existem várias estruturas diferentes
  - Do convencional acima

Overview da Re-expressão de Dados







Overview da Re-expressão de Dados

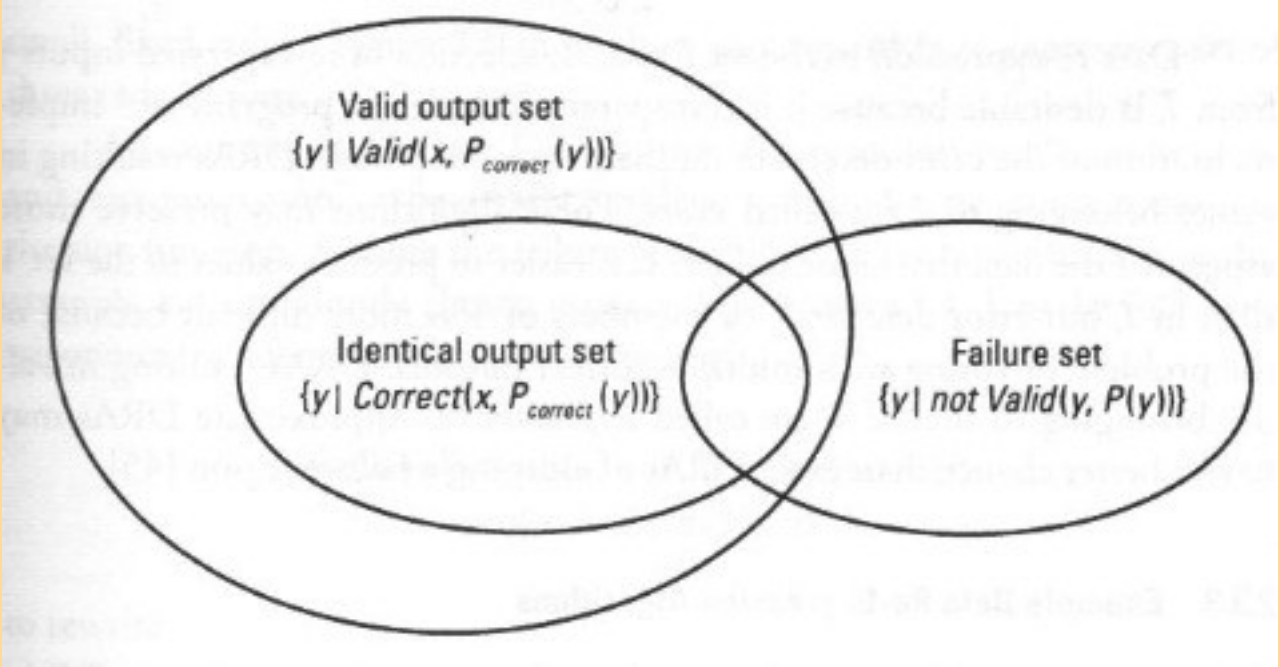
- a) *Basic re-expression method*
  - É o primeiro método proposto
- b) No segundo método:
  - Usa um método básico estendido
  - Permite criar um dado de entrada mais variado
- c) No terceiro método
  - permite múltiplas implementações dos algoritmos de re-expressão
  - O resultado de cada algoritmo é então combinado através de uma função com essa finalidade

Tipos de Saída

- Os requisitos dos DRAs podem ser definidos pelas características das saídas
- Existem três espaços de saída para um dado de entrada *x* e *y* (gerado pela re-expressão):
  - Conjunto de saída válido
    - Dado um *x* e *y* de entrada, eles geram saídas válidas
  - Conjunto de saída idêntico
    - Dado um *x* e *y*, eles geram saídas idênticas e corretas
  - Conjunto de falha
    - Dado um *y* de entrada, eles causa falhas

Conjunto de saída para um *x*

- Correct(*in*, *out*) é verdadeiro se e somente se *out* é uma saída válida para uma entrada *x* (original)
- $P_{correct}(x)$ produz uma saída correta de acordo com a especificação do sistema

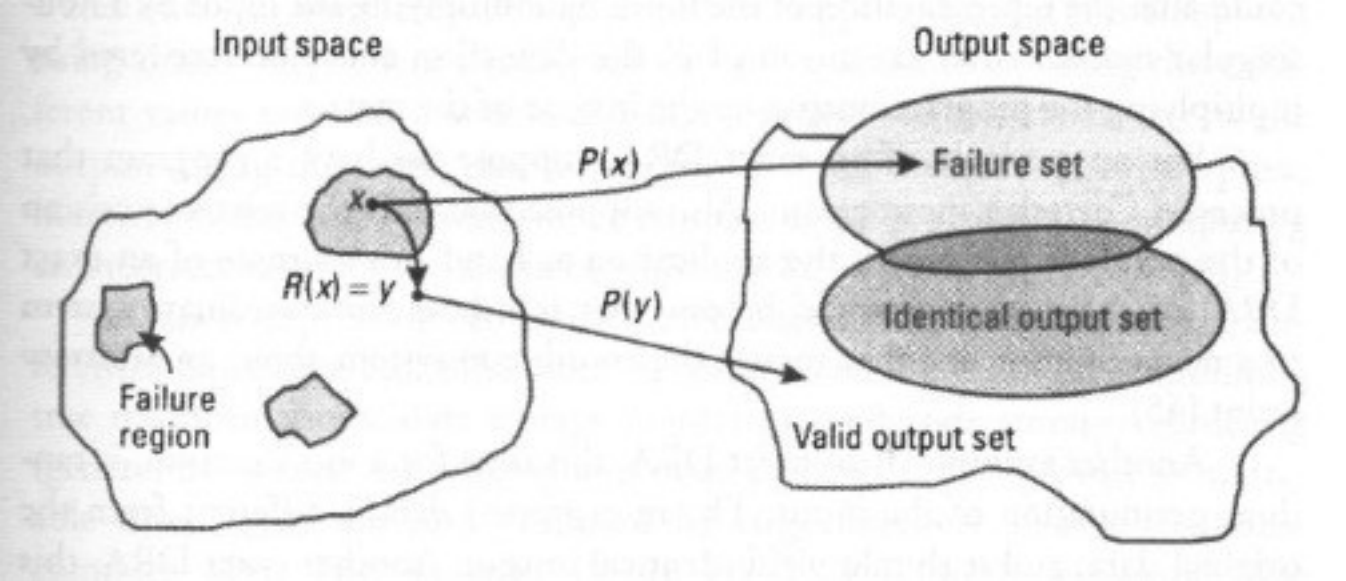


Conjunto de saída para um *x*

- O conjunto de saída *V* contém todos os *outputs* válidos ou aceitáveis dado um *x* como entrada
- Valid(*in*, *out*) é verdadeiro se e somente se *out* é válido ou aceitável dado um *x* como entrada
- O conjunto de falhas representa todos os *y* no qual um programa falha em prover um resultado
  - Aceitável dado um *x* como entrada
- O efeito da diversidade é medido pela proporção dos pontos *fora do conjunto de falhas*

DRA e Alteração de Valores

- DRA altera o valor de entrada original para prover uma saída válida

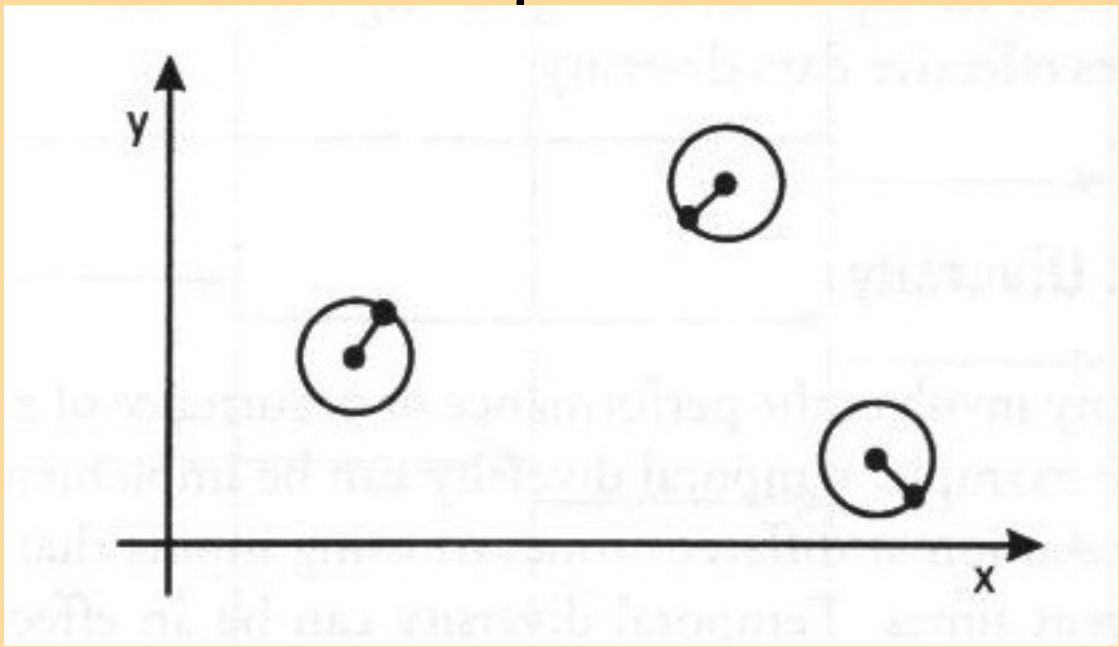


DRAs

- Re-expressão de dados no conjunto *I* (*Identical)* é desejável. Por que?
  - Erros são mais transparentes
- Dados derivados por DRAs no conjunto *I* são chamados como *exatos*
- É mais fácil produzir dados no conjunto *V* (válido)
  - Porém, é mais difícil lidá-lo pela necessidade de ter que escolher entre múltiplas saídas
- DRAs que geram valores no conjunto *V* são chamados *aproximados*
  - Podem ter mais chances de gerar saídas fora do conjunto *F*

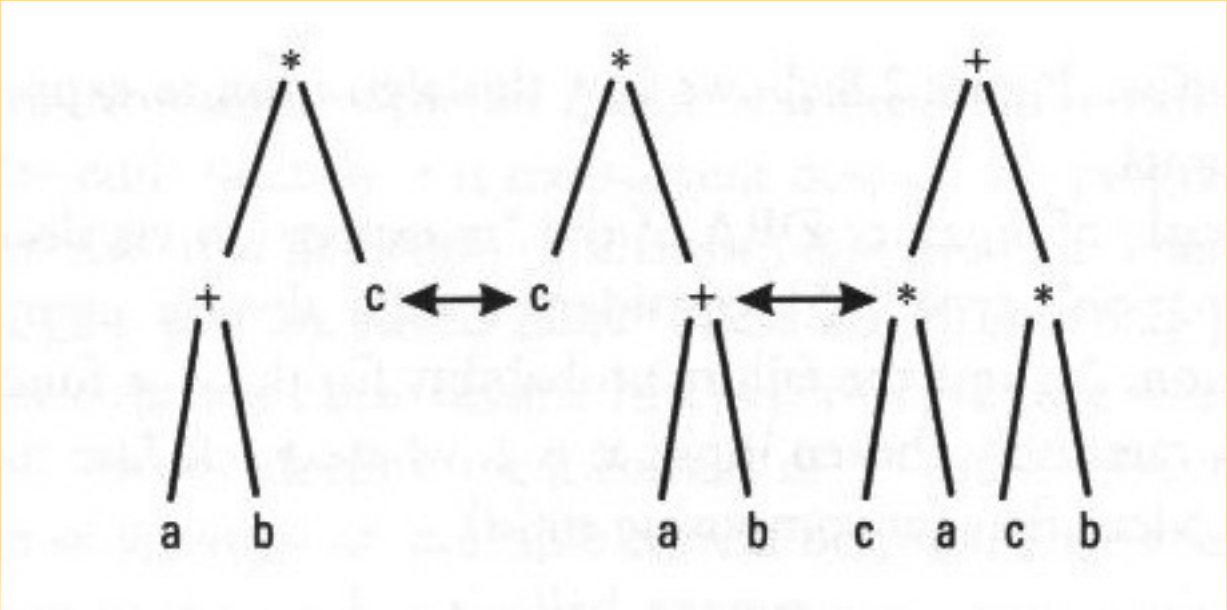
Exemplos de DRAs

- Em uma função Sort, um DRA pode subtrair cada valor de entrada (*postadjustment*)
- Leitura de dados sensores: uso de filtro de ruídos para permitir TFs
- Re-expressão de três pontos de radares



Exemplos de DRAs

- Uma simples transformação na árvore para avaliar a expressão *(a + b) * c*
- Três estruturas semânticamente idênticas
- A terceira é indesejável pela performance
- Mas, as diferentes representações podem fazer diferentes usos de registradores, por exemplo.



DRAs

- Os DRAs *exatos* podem ter alguns defeitos
  - Preservar precisamente aspectos de dados que levam a falhas
  - Levam todos os dados a serem re-expressados de forma a não sair da *região* de falhas
- DRAs aproximados podem dar maiores chances de saírem das *regiões* de falhas
- O desenvolvedor deve estudar e entender a aplicação com vistas a:
  - Prover um DRA que dê uma diversidade de dados eficaz

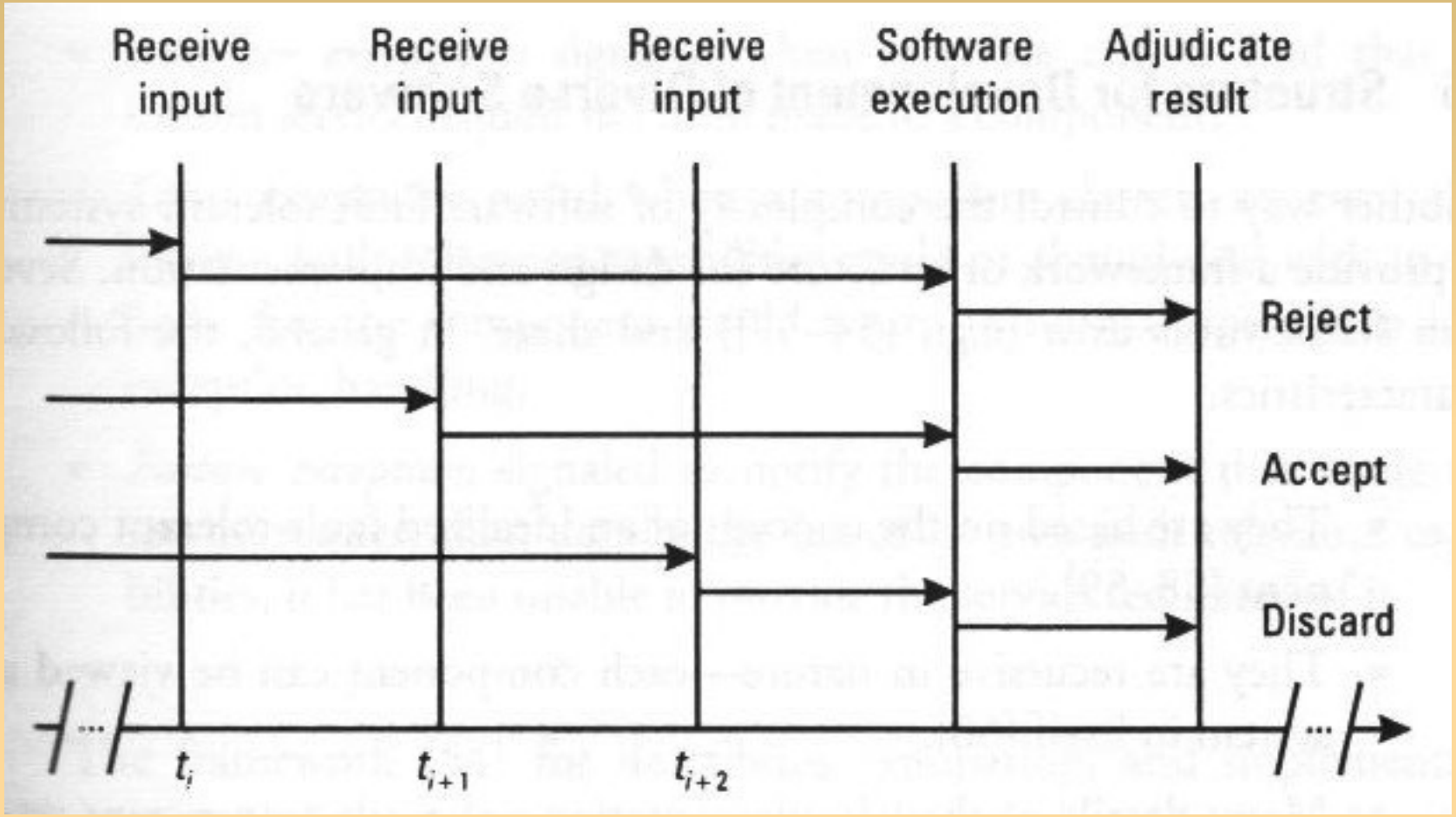
Diversidade Temporal

- O que é? Diversidade temporal envolve executar uma operação em períodos distintos
- Pode ser implementado de duas formas:
  - Re-executar a mesma operação em tempos distintos
  - Re-executar a mesma operação usando dados produzidos em tempos distintos
- Eficaz para lidar com falhas temporárias
- Diversidade temporal pode ser usado com a diversidade de dados

Diversidade Temporal

- Na próxima figura ilustramos um exemplo
- Um dado é recebido no tempo *t1*
- Ele é executado para produzir um resultado e este é avaliado pelo AT (*acceptance test*)
  - Exemplo: ele testa se é maior do que um valor máximo aceito
- Assumimos que o primeiro falha e o segundo passa o teste
- Se o nosso mecanismo de TF aceita três entradas, então o terceiro é descartado

Diversidade Temporal



- A diversidade temporal pode prover meios para implementar a diversidade de dados

Diversidade Temporal

- Os sistemas que precisam de TF são altamente complexos
- Para evitar tais falhas a primeira coisa é 'domar' a complexidade
- Uma das formas é estruturar o software em camadas e aplicar mecanismos de TF em cada uma delas
- Implementá-las usando a abordagem das *variantes*
- Outra forma de 'domar' esta complexidade é utilizar os *frameworks*

Frameworks de TFs

- Muitos dos detalhes de implementação são 'escondidos' dos desenvolvedores
- Eles dão uma interface bem definida para definir e implementar cada mecanismos de TF
- Possuem três componentes:
  - Controlador
  - Variantes redundantes
  - 'juiz' ou *adjudicator*
- Exemplos: Xu and Randell Framework

Frameworks de TFs

- Controlador
  - Orquestra as operções das técnicas de TF 'invocando' as variantes e usando um 'juiz' para determinar o correto funcionamento do sistema
- Variantes
  - Dão o mesmo serviço, porém através da diversidade de software e de dados
- 'Juiz' ou *adjudicator*
  - Seleciona um resultado 'provavelmente' correto dos obtidos pelas variantes

Então...

- Fim da parte de redundância
- Leiam até a Seção 2.6 do livro *Software Fault Tolerance Techniques and Implementation*