Usos da Adaptatividade em Simulação

João José Neto

9º WTA – janeiro de 2015

SIMULAÇÃO E SIMULADORES GUIADOS POR EVENTOS

Introdução: **simulação**

- É possível estudar o comportamento de um fenômeno qualquer através de diversos métodos, tais como, entre outros, a **observação** direta, a **modelagem matemática**, a **modelagem física** através de modelos mecânicos, elétricos, hidráulicos, etc., a **simulação**, analógica ou digital, etc.
- Convém lembrar que, ao contrário que muitos pensam, uma **simulação é sempre uma imitação**, sujeita a inúmeras restrições, nunca uma réplica do fenômeno simulado.
- A ***simulação*** de um fenômeno usando-se um **computador** digital consiste em utilizar a máquina para imitar, tão fielmente quanto possível, o fenômeno desejado.
- Isso pode ser feito executando-se um **programa** que implemente, através de sua **lógica**, um **modelo computacional** do fenômeno em questão.

Modelos determinísticos

- Há modelos computacionais de diversas naturezas, que utilizam diferentes técnicas para recriar o comportamento do sistema simulado:
  - modelos analíticos, ***determinísticos*****,** baseados em **fórmulas** e **equações** matemáticas que avaliam, de forma tão realística quanto possível, o comportamento do fenômeno frente a estímulos
  - **modelos *estocásticos***, baseados em métodos **estatísticos**, que estudam o fenômeno observando seu comportamento **probabilístico**.
- Em geral, ***modelos determinísticos*** reagem sempre da mesma maneira a um dado estímulo, por serem, em geral, regidos por **funções analíticas**, que priorizam o cálculo de **respostas específicas** para estímulos determinados, ou seja, para os casos particulares que forem sendo estudados, sem fornecer diretamente informes globais acerca de seus comportamentos.

Modelos estocásticos e híbridos

- Os ***modelos estocásticos***, por sua vez, tendo sua fundamentação em **métodos estatísticos**, utilizam-se de **sorteios**, **distribuições** de probabilidades, dados **paramétricos** e outros, que privilegiam a informação acerca de **tendências**, **médias**, comportamentos **típicos** e assim por diante, em detrimento da atenção a casos individuais.
- Frequentemente, a **utilização conjunta** das duas técnicas parece fornecer resultados mais aproveitáveis, através de ***modelos híbridos*****,** nos quais são estabelecidas formulações **determinísticas** apenas para os elementos simulados cujos **parâmetros individuais** tenham real interesse para a simulação realizada, escolhendo-se modelos **estatísticos** para elementos globalmente relevantes, mas cujo comportamento individual não seja essencial ao estudo realizado.

Sistemas reativos

- ***Sistemas reativos*** formam uma classe de sistemas de grande importância em computação.
- Trata-se de sistemas que recebem, como **estímulos**, ocorrências ou sinais oriundos de **fontes** tanto **externas** como **internas** ao sistema, e que **reagem** a esses estímulos de forma **seletiva**, gerando para cada sequência de estímulos recebida pelo sistema uma sequência correspondente de reações.
- Podemos definir ***eventos*** como sendo quaisquer ocorrências (tipicamente, mas não obrigatoriamente, aleatórias e assíncronas) que sejam de interesse para a compreensão do sistema em estudo.
- São fatos que, no sistema em análise, **ocorrem** em **instantes** que podem ser: **ou conhecidos**, **ou calculados**, **ou estimados**.

Simulação por eventos

- Uma importante técnica, utilizada para efetuar simulações de sistemas reativos, denomina-se ***simulação dirigida (ou orientada) por eventos***.
- Neste tipo de simulação, um **modelo** do fenômeno estudado **imita** o comportamento do sistema, sendo representado através de um **programa** de computador.
- Isso é feito através de uma **lógica,** que estima, para cada possível cenário em que se encontre o sistema, a sua **resposta aos eventos ocorridos** (ou seja, suas **reações aos estímulos** em questão).

Eventos independentes

- São ditos ***eventos independentes*** aqueles cuja **ocorrência** natural **independe do sistema** que se está considerando, e que, em programas de simulação, têm sua ocorrência arbitrariamente estipulada para **acontecer em instantes preestabelecidos**.
- Isso ocorre, por exemplo, com o acionamento de botões de controle, com o instante de chegada de programas ao sistema, com o instante imposto pelo programador para finalizar a simulação, com o instante em que é ligado inicialmente o relógio de tempo real da máquina simulada, e muitos outros da mesma natureza.

Eventos dependentes

- São chamados ***eventos dependentes*** todos os eventos cujo instante de ocorrência **o simulador é capaz de calcular ou de estimar**.
- É o caso, por exemplo, do momento de término de uma operação de entrada ou saída de dados (calculável a partir do instante de acionamento e do tempo estimado para a conclusão da operação), o próximo instante de interrupção do relógio de tempo real (calculável a partir do instante do seu último acionamento e do ciclo do relógio) etc.

ASPECTOS DE PROJETO

Projeto

- O **projeto e a implementação de um *simulador dirigido por eventos***, tem como meta estimar respostas do sistema simulado a estímulos recebidos do meio externo, na forma de uma lista representando **todos os eventos independentes** que deverão alimentá-lo.
- Antes de iniciar sua operação, esse programa deverá efetuar uma **entrada de dados**, através da qual deverá informar-se sobre cada um dos eventos independentes que deverão estimulá-lo, bem como os instantes impostos para a sua ocorrência na simulação.
- Como sistema reativo, o sistema simulado deverá executar uma **rotina de tratamento** para cada uma dessas ocorrências, e, como resultado da execução dessas rotinas de tratamento, são produzidas, finalmente, as **saídas simuladas** do sistema em estudo.

Lista de eventos

- Esses **dados** constituem a versão inicial de uma ***lista de eventos programados,*** a serem tratados pelo simulador, lista essa que sofrerá, durante a simulação, **alterações** impostas pelo próprio simulador, conforme descrito a seguir.
- A lista de eventos geralmente é implementada na forma de uma **lista ligada**, que deve ser mantida **ordenada** em ordem **crescente**, de acordo com o **instante de ocorrência previsto** para os seus diversos eventos.
- Em caso de **conflito** (quando dois ou mais eventos estiverem programados para ocorrer **simultaneamente**), pode-se adotar, como **política de desempate,** que a **ordem de inserção dos eventos na lista** deve prevalecer , ou seja, deve ser tratado em primeiro lugar o evento que tiver sido inserido antes na lista de eventos programados.

Lógica da simulação

- Construída a lista inicial de eventos (compreendendo apenas eventos independentes), a simulação propriamente dita pode ter início, e para isso o simulador entra em um **ciclo de execução** no qual:
  - **um evento é retirado, do início da lista** de eventos, para ser tratado pelo simulador
  - se o **instante corrente de simulação** for **posterior** àquele previsto para a ocorrência do evento, este deverá ser simulado **como se tivesse ocorrido exatamente no instante para ele previsto**
  - caso contrário, sua ocorrência deverá ser simulada **no instante corrente de simulação**.
  - **identifica-se o tipo de evento**, e aciona-se a correspondente rotina de tratamento, de forma análoga ao que é executado pelo hardware quando da interpretação de uma instrução de máquina, ou da ocorrência de um pedido de interrupção.
  - Executada a rotina de tratamento do evento, **retorna-se ao primeiro passo**, enquanto **houver eventos** a serem tratados **e não for atingido o instante final** especificado de simulação.

Simulador estocástico de processamento multiprogramado, dirigido por eventos

- Um simulador estocástico dirigido por eventos pode então ser construído com base em uma lista inicial de eventos independentes, adequada para ativar todas as suas funções internas. Por exemplo:
  - o **instante inicial de acionamento do relógio** de tempo real,
  - o **instante final de simulação** e
  - os **instantes de chegada** de cada um dos elementos de um conjunto de programas independentes, para cada qual esteja especificado, por exemplo:
    - o tempo total de processamento,
    - a quantidade estimada de memória de que necessitará e
    - o número estimado de operações de entrada/saída que deverá realizar.

- Como saída, o simulador deve gerar uma lista que relate todos os instantes previstos e executados para todos os eventos processados, na simulação de cada um dos programas.
  O formato pode ser, por exemplo, o seguinte:

  <instante> <tipo de evento> <identificação do programa> <ação executada> <resultado obtido>

onde:

- **<instante>** é o valor corrente do relógio do simulador (**instante corrente de simulação**)
- **<tipo de evento>** indica qual foi a **ocorrência** observada neste instante de simulação
- **<identificação do programa>** indica **qual dos programas** simulados provocou o evento
- **<ação executada>** indica **qual rotina foi executada** como resposta do sistema ao evento
- **<resultado obtido>** indica os **efeitos dessa reação** sobre a situação corrente do programa simulado

Esse procedimento deve ser aplicado a diferentes listas de evento iniciais.

Os resultados da simulação devem então ser analisados, comparados, comentados.

ASPECTOS DE IMPLEMENTAÇÃO

Informações relevantes

- **Instantes inicial e final** de simulação;
- Dados sobre os **jobs a serem simulados**;
- **Estímulos programados** para o simulador, e respectivos **instantes previstos de ocorrência**;
- **Modelos** para os vários elementos a simular:
  - Memória e suas partes;
  - Jobs / Processos;
  - Discos e outros Dispositivos de Entrada / Saída;
  - Arquivos, Mensagens e Sistema de Comunicação;

Evento = (tipo, instante)

- ***Tipo*** – codifica algum fenômeno associado;
- ***Instante*** – prevê seu momento de ocorrência;
- ***Independentes*** – impostos pelo usuário. Ao início da simulação, só se conhecem eventos independentes;
- ***Dependentes*** – calculados pelo simulador. Uma **lista de eventos**, mantida em ordem crescente de *instante* pelo simulador, abriga todos os eventos, dependentes ou não, que ainda não tenham sido tratados;

Loop básico de simulação

Enquanto houver eventos a serem simulados:

- Determinar o próximo evento a ser tratado;
- Executar o tratamento correspondente ao tipo do evento em questão:
  - Evento do tipo 1: rotina de tratamento 1;

    ...
  - Evento do tipo n: rotina de tratamento n;
- Coletar informações de controle da simulação;
- Exibir informação de acompanhamento;

Consolidar e imprimir as informações obtidas;

“Traduzindo”:

Enquanto a lista de eventos não estiver vazia:

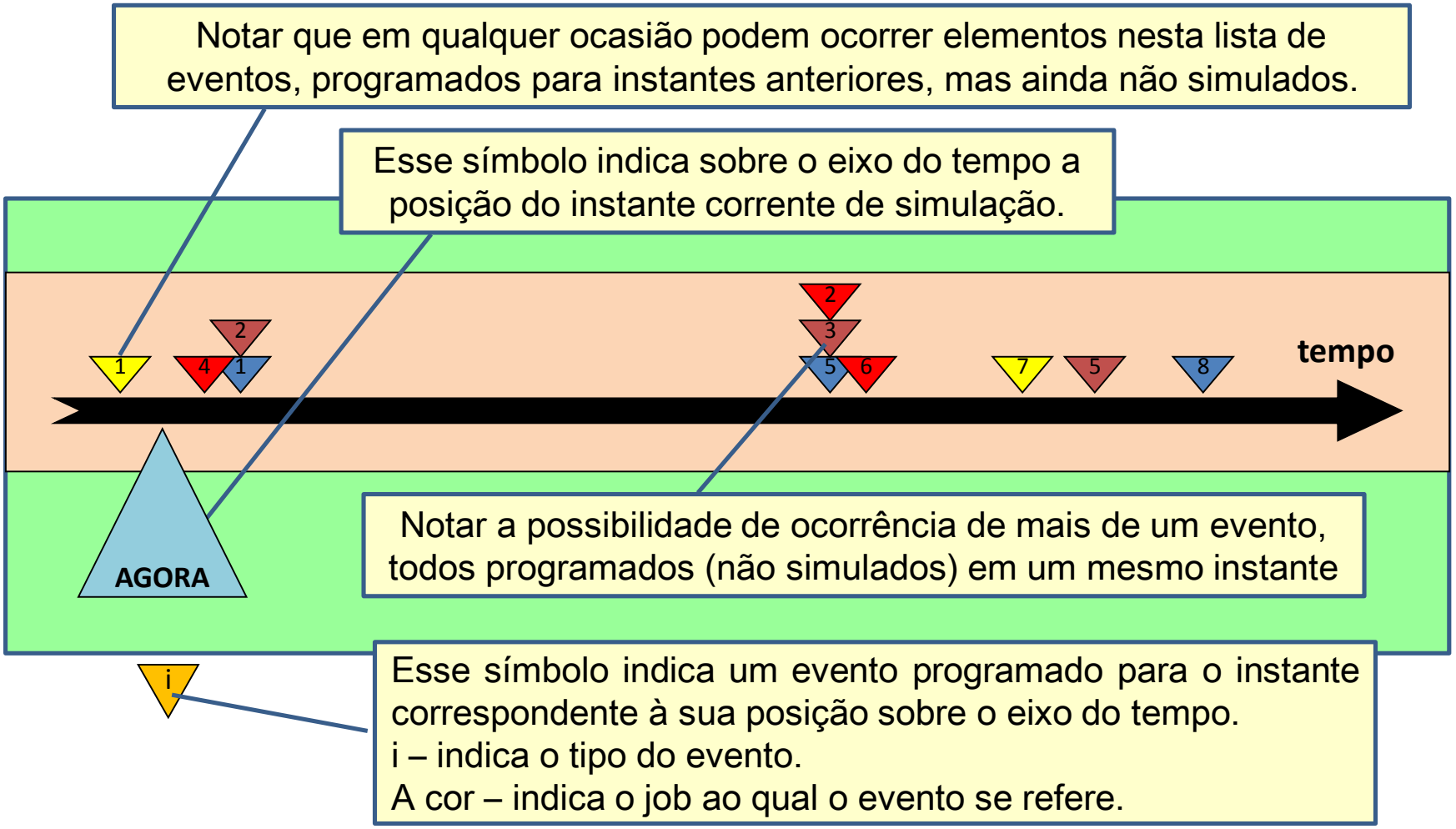
- Extrair da lista de eventos o primeiro da lista, ou seja, aquele que estiver programado para ocorrer mais cedo; seja ele um evento (i,t) do tipo i, programado para ocorrer no instante t de simulação.
- Tratá-lo da mesma forma como um sistema operacional trataria uma interrupção: executando a i-ésima rotina de tratamento extraída de um vetor de interrupções
- Coletar informações de controle da simulação;
- Imprimir informação de acompanhamento;

Consolidar, formatar e apresentar as informações colhidas no processo de simulação.

Simulação do tempo

- AGORA = instante de simulação corrente;
- Lista de eventos = lista de pares ordenados, da forma (*tipo*, *instante*), em ordem crescente de seu *instante* de ocorrência;
- Por convenção, o último evento dessa lista deve ser obrigatoriamente do *tipo* “fim de simulação”.
- Para determinar o próximo evento a tratar, basta extrair o primeiro elemento da lista.
- Então atualiza o instante de simulação corrente para o maior dentre os valores de AGORA e do *instante* do evento;

Lista de eventos programados



Observação sobre o tempo

- No mundo real é possível considerar que o tempo flui linear e continuamente, constatando-se situações com longos períodos sem atividade;
- Em simulações discretas, como as dirigidas por eventos, é desnecessário simular os períodos de inatividade, podendo-se poupar processamento desprezando-se tais períodos sem efetuar ação alguma de simulação;
- Assim, o valor da variável AGORA evoluirá de forma discreta, saltando de um valor para outro ao longo da simulação, de acordo com os *instante*s previstos de ocorrência dos eventos da lista.

Simulação dos eventos

- Extraído no instante de simulação AGORA um evento (i, t), executa-se a rotina de tratamento i, que deve efetuar toda a computação responsável pelas atividades ligadas a ocorrências de eventos do tipo i;
- Se t < AGORA, interpreta-se que não foi possível ao simulador atender o evento no instante previsto para a sua ocorrência, por isso seu atendimento foi postergado para o instante AGORA.
- Entre outras atualizações das estruturas de dados de simulação, deve ser atualizado o novo valor do instante corrente AGORA.
- Essa atualização não precisa ser contínua, podendo saltar os intervalos sem atividade.

PROPOSTA INTUITIVA

Funcionamento de máquinas abstratas programáveis.

- Máquinas abstratas programáveis operam tipicamente sobre um conjunto de **elementos de memória**, que contêm **dados** e o **código** referente ao programa a ser executado.
- Este programa está denotado na forma de **instruções codificadas** em um formato predeterminado, de acordo com um conjunto de operações elementares específico da máquina abstrata em questão.
- Para **cada instrução** contida no código do programa em execução, um **conjunto apropriado de operandos é determinado**, e sobre eles **uma dessas operações é executada**.
- Após a execução de cada instrução, **determina-se o endereço da próxima instrução a executar** e reinicia-se o processo, até que a máquina entre no estado "**halt**".

Um transdutor converte dinamicamente programas em sequências de operações

- Um autômato pode aceitar um código objeto absoluto e carregá-lo na memória do processador ("***Loader***").
- Em função dos valores dos dados, as instruções do código objeto devem ser correspondentemente interpretados.
- Um transdutor ("***Virtual machine interpreter***"), baseado no autômato inicial, converte a imagem da memória do processador em uma sequência de operações incondicionais a serem executadas
- Finalmente, um interpretador (na forma de um conjunto de rotinas de tratamento dessas operações , similares às de atendimento de interrupções) executa a sequência de ações especificadas na sequência de operações na qual a imagem da memória foi convertida pelo transdutor.

Esquema de um ambiente síncrono, não-programável, acionado por botões

Acionamento manual por painel de botões

Aplicação:

Execução de operações manualmente acionadas por um painel.

Esta é a situação típica de uma máquina de calcular não programável, ou a de uma interface de operação de algum controlador que utilize como entrada um painel acionado por botões de controle.

Não há um programa a ser executado, então o operador, pressionando os botões da interface, aciona manualmente a execução das correspondentes operações elementares do ambiente.

A cada operação acionada pelo aperto de um botão, a operação correspondente é executada sobre dados contidos em locais preestabelecidos da máquina (virtual), obtendo os resultados da operação, e depositando-os em locais predefinidos.

Não sendo programáveis, dispositivos desta categoria naturalmente exigem que o operador explicite todos os passos que deseja executar. Assim, passo a passo, as operações elementares efetuadas vão realizando o processamento desejado sobre os dados manipulados.

Esquema de um ambiente síncrono programável, para software sequencial

Acionamento síncrono, uniprogramado

Aplicação:

Execução de scripts codificados na linguagem da máquina abstrata.

Por questão prática, o programa pode ser recebido na forma de um código-objeto adequadamente empacotado para fins de segurança e confiabilidade.

Neste caso, um carregador (loader) se faz necessário para desempacotá-lo e criar na memória uma imagem do conjunto de pacotes que compõem o programa-objeto.

Nesta situação, todos os programas devem estar codificados na linguagem de uma mesma máquina virtual.

Uma vez na memória, sabendo-se o endereço de execução, ocorre um processo de sequenciação, no qual a meta-sequência, que é o programa, é instanciada em função dos dados correntes, produzindo-se a sequência propriamente dita de operações a serem executadas.

A sucessiva execução dessas operações constitui a interpretação final do programa.

Esquema de um ambiente programável assíncrono, para software concorrente

Acionamento assíncrono, multiprogramado e multiprocessado

Aplicação: Execução simultânea de diversos scripts (eventualmente heterogêneos) codificados cada qual na linguagem de sua própria máquina abstrata.

Na memória do respectivo processador virtual, cada programa tem seus próprios espaços de endereçamento e seus próprios endereços de execução, podendo estar codificado cada qual na linguagem da sua própria máquina virtual.

É importante lembrar que para este estudo é irrelevante se as máquinas virtuais são implementadas em um ou mais hardwares físicos.

Sabendo-se o endereço de execução de cada programa, é executado um processo de sequenciação, no qual o programa, que é uma meta-sequência, é instanciado, em função dos dados correntes, obtendo-se a correspondente sequência propriamente dita das operações elementares incondicionais a serem executadas.

A interpretação final do programa é constituída pela execução dessa sequência de operações.

Combinações prováveis de uso

- Os slides anteriores mostram que todos os esquemas apresentados podem ser unificados, de forma que um mesmo programa-mestre possa ser utilizado para diferentes simulações, bastando para isso configurá-lo segundo as necessidades em cada situação, conforme resumido nesta tabela. (MCP=máquina de calcular programável)

Acionamento / Utilização	Botões	Instruções	Mensagens síncronas	Mensagens assíncronas
Manual .	Operação com acionamento de botões apenas	MCP: **Entrada de programa** (modo programação)	(só envio)	(só envio)
Programada .	MCP: **Entrada de dados** (modo execução) ou **de programa** (modo programação)	MCP: **Entrada de código objeto de programa** (modo programação)	Objetos/ processos comunicantes em um mesmo programa	Processos comunicantes por mensagens em um mesmo programa
Concorrente .	MCP: **Entrada de dados** (modo execução)	MCP: **Entrada de dados** (modo execução) ou de **código objeto de programa** (modo programação)	Objetos/ processos comunicantes envolvendo vários programas	Processos comunicantes por mensagens envolvendo vários programas

Deste estudo conclui-se

- Há fortes **paralelos** entre diversos processos de **simulação** e a operação de **máquinas abstratas**.
  - Máquinas abstratas representam linguagens, portanto podem ser formalmente **especificadas gramaticalmente**, podendo-se de tal gramática obter automaticamente um simulador equivalente.
  - Os paralelos encontrados autorizam-nos a **formalizar gramaticalmente o simulador** desejado:
    - Especifica-se manualmente uma **gramática** que formalize a linguagem referente à máquina abstrata desejada
    - Gramáticas podem ser, fácil e automaticamente, **convertidas em reconhecedores** (máquinas abstratas) equivalentes.
    - A **simulação dessa máquina abstrata** equivale à operação do reconhecedor assim obtido.

Proposta do processo

- **Especificar gramaticalmente uma linguagem** que formalize o comportamento desejado do simulador.
- **Gerar** (automaticamente) uma **máquina abstrata** equivalente, bem como seu **simulador**.
- **Preparar dados** de entrada para suprir o simulador obtido, **instanciando** assim **a simulação** desejada.
- **Simular** computacionalmente a máquina abstrata e **colher os resultados de simulação** obtidos com esses dados.

Quando empregar adaptatividade?

- Em situação nenhuma o uso da adaptatividade é obrigatório , devendo ser ela livremente utilizada, a critério do projetista.
- Caso a caso, podem haver condições possivelmente indicativas de (des)vantagens no seu uso, devendo elas ser investigadas, justificando a sua adoção ou rejeição.
- A adaptatividade deve ser considerada recomendável quando for possível aferir matematicamente potenciais vantagens expressivas, que decorram de sua utilização.
- Por outro lado, convém evitar seu uso caso se constatem potenciais desvantagens no seu emprego ou não se encontrem indicativos de possíveis benefícios.

Em que situações é oportuno utilizá-la?

- Em nenhuma situação é imprescindível o uso da adaptatividade, pois a Máquina de Turing [versão formal de algoritmo] é auto-suficiente para exprimir qualquer algoritmo, dispensando pois a adaptatividade.
- Tecnicamente, portanto, o uso dessa técnica só se justifica se for possível **constatar alguma superioridade evidente em relação às técnicas usuais**: em custo, eficiência, expressividade, clareza, concisão etc.
- Para isso, é desejável um estudo formal, quantitativo, do comportamento comparativo dos parâmetros de interesse, em relação a um equivalente clássico referencial, para validar propriamente a vantagem da incorporação da adaptatividade a um projeto.

Em que situações é contraindicada?

- Após um **estudo quantitativo** (provavelmente baseado na teoria da **complexidade computacional**) dos parâmetros de desempenho considerados significativos, e de sua **comparação** a casos clássicos correspondentes, nos quais a **funcionalidade seja preservada, sem contudo empregar a adaptatividade**, pode-se considerar inadequado adotar a adaptatividade em um projeto se os índices de mérito estudados se mostrarem:
  - Relativamente **similares, ou inferiores**.
  - Relativamente **similares, ou ligeiramente superiores**, devendo-se no caso priorizar o **custo adicional do aumento da complexidade** devido à inclusão da adaptatividade.
  - Superiores, porém **apenas em casos pouco expressivos**

Como a adaptatividade pode ser propriamente usada em um ambiente de simulação?

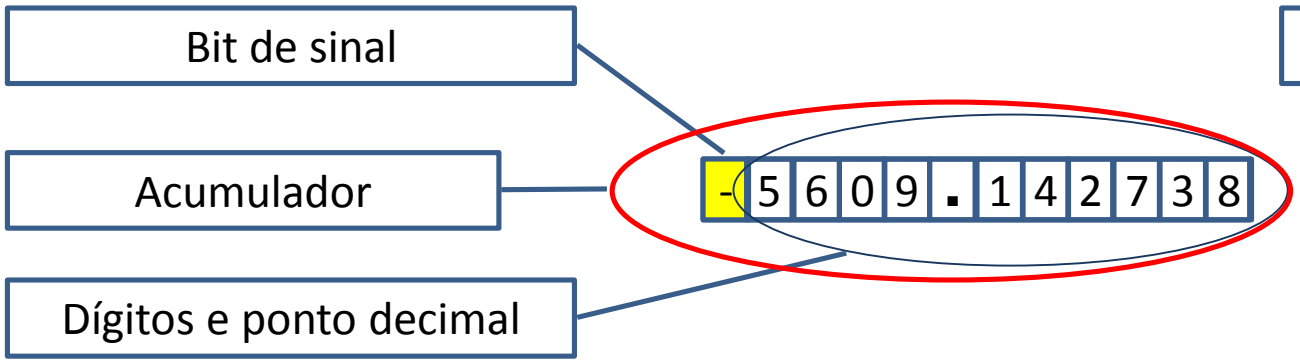
- Na **modelagem** de fenômenos suficientemente heterogêneos, mutáveis e complexos, tais como:
  - Processos de decisão e de classificação
  - Escaladores de eventos, sequenciadores
- Como **recurso de programação**, sempre que for considerado conveniente. Por exemplo:
  - “Margaridas” para representar processos combinatórios
  - Autômatos adaptativos, para topologias variáveis

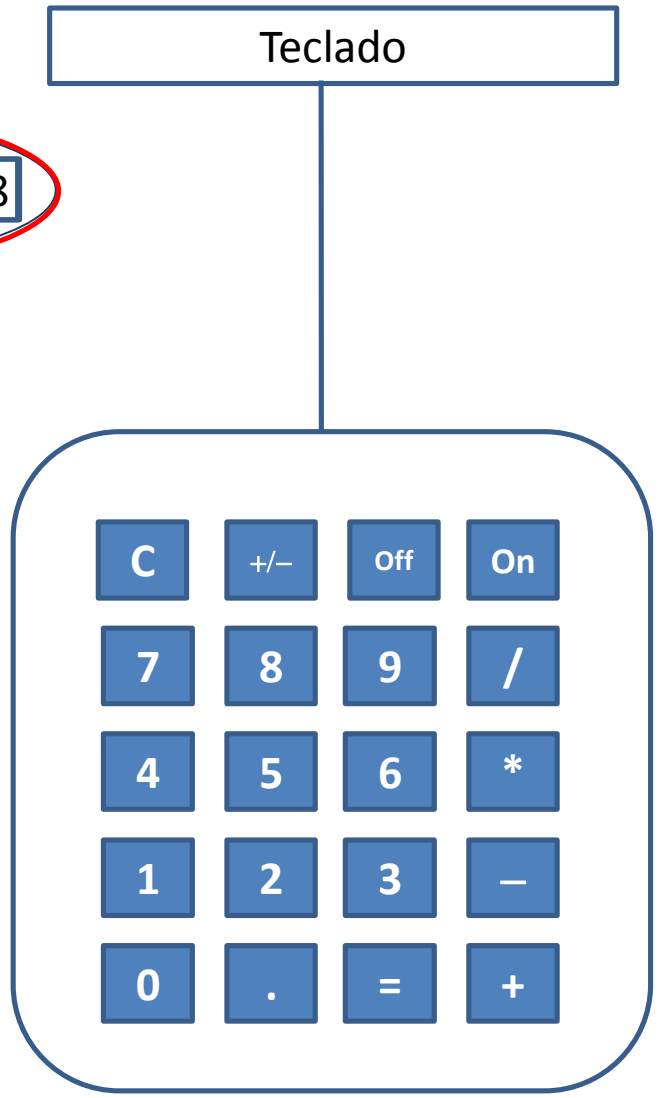
APLICAÇÕES

Visão de um hardware programável (= computador ou similar) como um dispositivo abstrato que pode portanto ser gramaticalmente formalizado logo convertido (automaticamente) em uma máquina de estados para efeito de simulação, posteriormente executada (também automaticamente) em um ambiente (gerado automaticamente) que ao ser executado interpreta (automaticamente) os dados de especificação das tarefas a processar (fornecidos como dados de entrada, os eventos na forma de apertos de teclas da máquina de calcular).

APLICAÇÃO 1 – SIMULAÇÃO DE MÁQUINA DE CALCULAR DE 4 OPERAÇÕES

Máquina de calcular – 4 operações





Eventos do teclado	Ação da calculadora
C	Clear limpa o acumulador
+/-	troca o bit de sinal do acum.
On	liga e limpa a calculadora
Off	desliga a calculadora
Dg (0..9)	insere próximo dígito no acum.
Ponto	insere ponto decimal no acum.
Igual	finaliza operação iniciada
Op (+, –, *, /)	inicia operação com o acum.

Especificação sintática

M = “On” { X } “Off” .

X = “C” | “=” | “+/–” | N { “=” } { Op N { “=” } } .

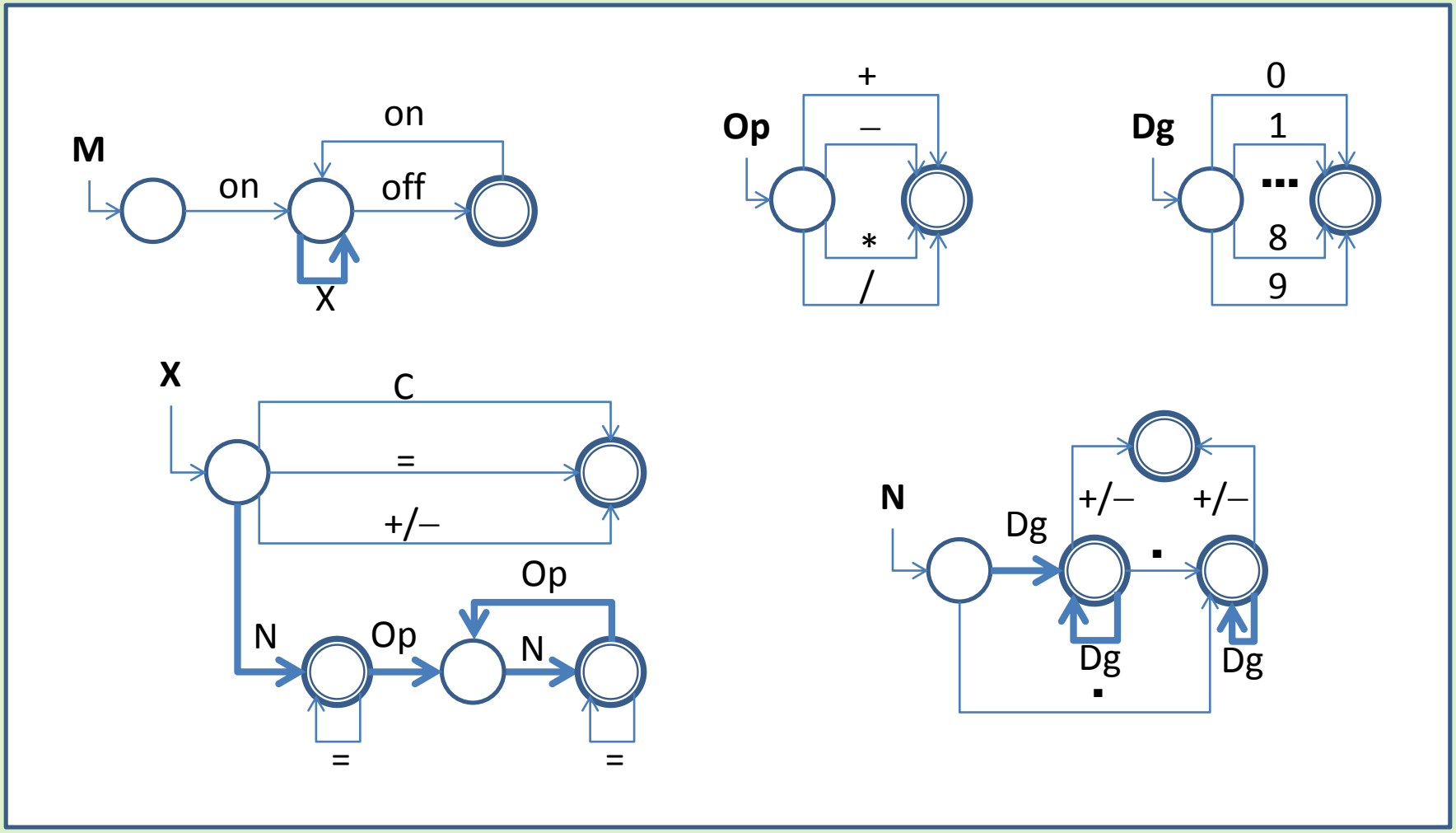
N = (Dg { Dg } [“.” { Dg }] | “.” { Dg }) “+/–” .

Op = “+” | “–” | “*” | “/” .

Dg =“0”|“1”|“2”|“3”|“4”|“5”|“6”|“7”|“8”|“9”.

Reconhecedor resultante

- A partir desta gramática, obtém-se facilmente o autômato de pilha estruturado abaixo:



Lógica do programa reconhecedor

- Entrada: sequência de teclas digitadas, cada qual podendo ser uma das que constam na tabela fornecida.
- Saída: máquina parada, com o resultado, da última operação realizada, no acumulador.
- Lógica:

LOOP: Leitura de uma tecla

Execução do tratamento correspondente à tecla

Go to LOOP

(Continuação)

- Tratamento das teclas (qualitativo, simplificado):
  - C (Clear) zera o acumulador
  - +/– (troca sinal) faz acumulador := – acumulador
  - On(liga e limpa) zera o acumulador e limpa pendências
  - Off(desliga) entra em estado desligado
  - Dg (0..9) concatena dígito ao acumulador
  - . concatena ponto decimal ao acumulador (se legítimo)
  - = completa a operação pendente
  - Op (+, –, *, /) torna pendente a operação digitada

Lógica do simulador convencional

LOOP: Leitura de uma tecla
De acordo com a tecla digitada executar uma das seguintes rotinas:

Se C : { Acumulador := 0.0; }
Se +/−: { Acumulador := − Acumulador; }
Se On: { Acumulador := 0.0; Parcial:= 0.0;
pendente:=false; desligado:= false; ponto:=false; }
Se Off: { desligado:= true; }
Se 0..9: { Acumulador := Acumulador && dígito; }
Se . { if ponto then erro else { Acumulador := Acumulador && “.” ; ponto:=true; }
Se = { Acumulador := pendente (Parcial, Acumulador);
Parcial:=0.0; pendente:= false; }
Se +, −, *, /: { Parcial:=Acumulador; Acumulador:= 0.0; pendente:= operação digitada; }

Go to LOOP

Alguns possíveis usos da adaptatividade

1. Como filtro de entrada, **para corrigir digitação incompatível com a sintaxe**: em todos os estados do autômato a detecção de símbolos não previstos pode provocar a emissão de algum informe de erro, podendo ser ignorada ou então finalizar a sequência que estiver sendo processada.

2. Na entrada dos numerais, **complementando a sintaxe livre de contexto**, pela incorporação da contagem de dígitos significativos, do tratamento de zeros à esquerda, da entrada de mais de um ponto decimal, de sequências de teclas de operação sem a entrada de operandos, e outras situações similares.

3. No corpo da lógica do simulador, há um grande comando de decisão múltipla. Em simuladores mais complexos este comando pode ficar muito grande, sendo que boa parte dos casos raramente ou nunca são ativados durante uma particular simulação.

Um ganho pode se obtido **inserindo-se adaptativamente os códigos** que tratam os diversos casos, **estritamente se estes forem de fato ativados**.

Para isso, basta iniciar com todos os casos sem tratamento especificado, e associar uma exceção aos casos que nunca foram ativados. O tratamento desta exceção insere adaptativamente o código propriamente dito do tratamento da tecla, desativando futuras exceções para aquele caso.

4. Desejando-se fazer um simulador que seja capaz de **alterar sua funcionalidade** (por exemplo, para uso e experimentação em laboratório) a adaptatividade pode ser empregada com propriedade, acrescentando-se ou eliminando-se partes do código, de acordo com a modificação de funcionalidades desejada.
Por exemplo, se desejarmos que a calculadora passe a utilizar um teclado mais amplo, novas funções poderão ser incorporadas de forma muito similar ao que foi feito anteriormente. (Ver próximo slide)

Esta extensão não foi desenvolvida aqui por não trazer novidades conceituais.

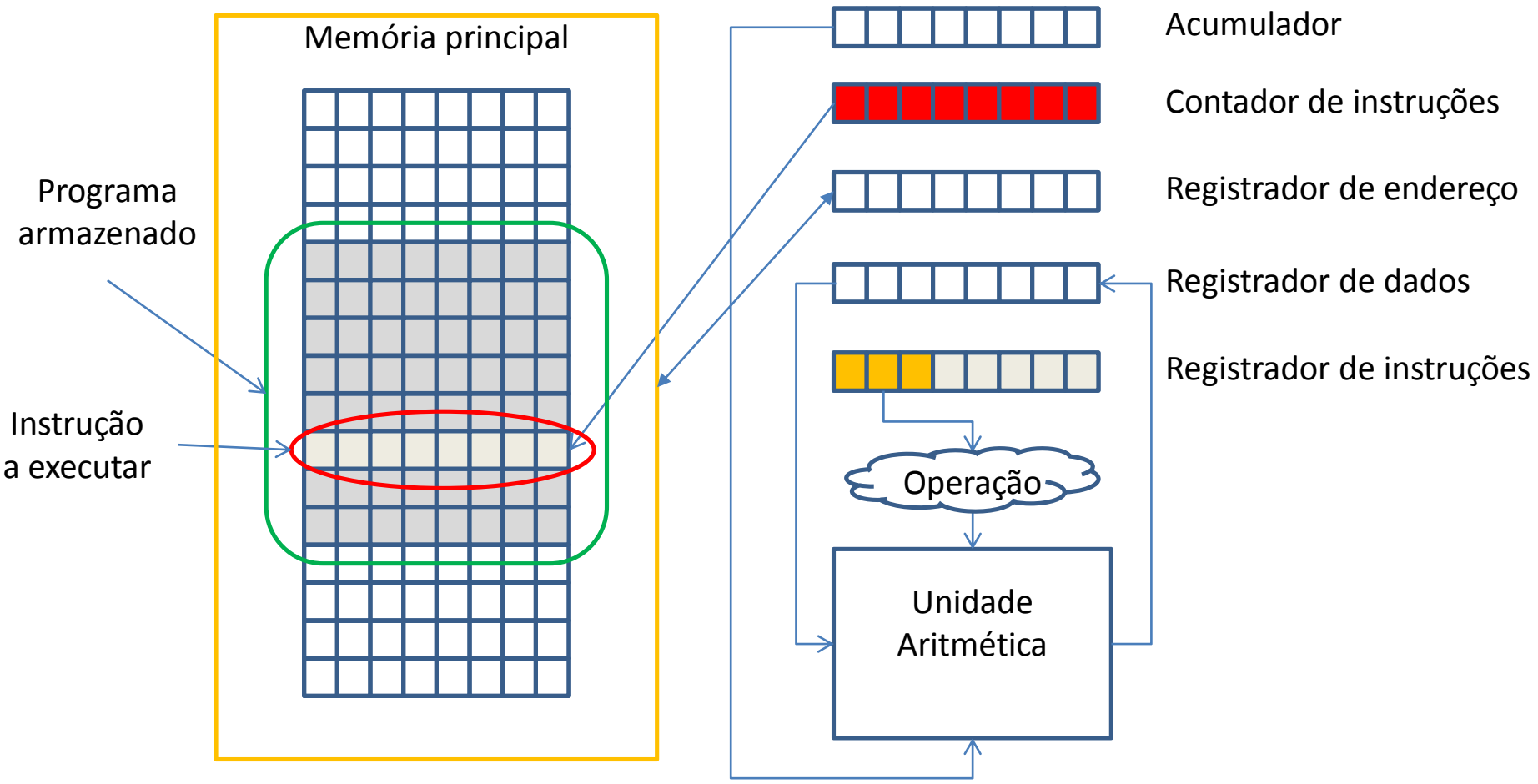
Visão de um hardware programável (= computador ou similar) como um dispositivo abstrato que pode portanto ser gramaticalmente formalizado logo convertido (automaticamente) em uma máquina de estados para efeito de simulação, posteriormente executada (também automaticamente) em um ambiente (gerado automaticamente) que ao ser executado interpreta (automaticamente) os dados de especificação das tarefas a processar (fornecidos como dados de entrada, os eventos assumem a forma de instruções de máquina introduzidos inicialmente na forma de um programa em linguagem de máquina a ser executado).

APLICAÇÃO 2 – SIMULAÇÃO DE UM PROCESSADOR PROGRAMÁVEL

Processador programável

- Computador ou similar – p.ex., máquina de Von Neumann
- Dispositivo abstrato, pode ser gramaticalmente formalizado
- Pode ser convertido (automaticamente) em máquina de estados para efeito de simulação
- Esta máquina pode ser inserida em um ambiente (gerado automaticamente)
- Pode ser também posteriormente interpretada (automaticamente)
- Ao ser executado interpreta (automaticamente) os dados de entrada. que especificam tarefas a processar
- Os eventos (estímulos) assumem a forma de instruções de máquina (que constituem os programas a serem executados, em linguagem binária, de máquina)

Processador típico



Especificação sintática

C = { P | D } X .

P = “P” “=” B .

D = “D” “=” B .

B = { N “:” N { “,” N } “;” } .

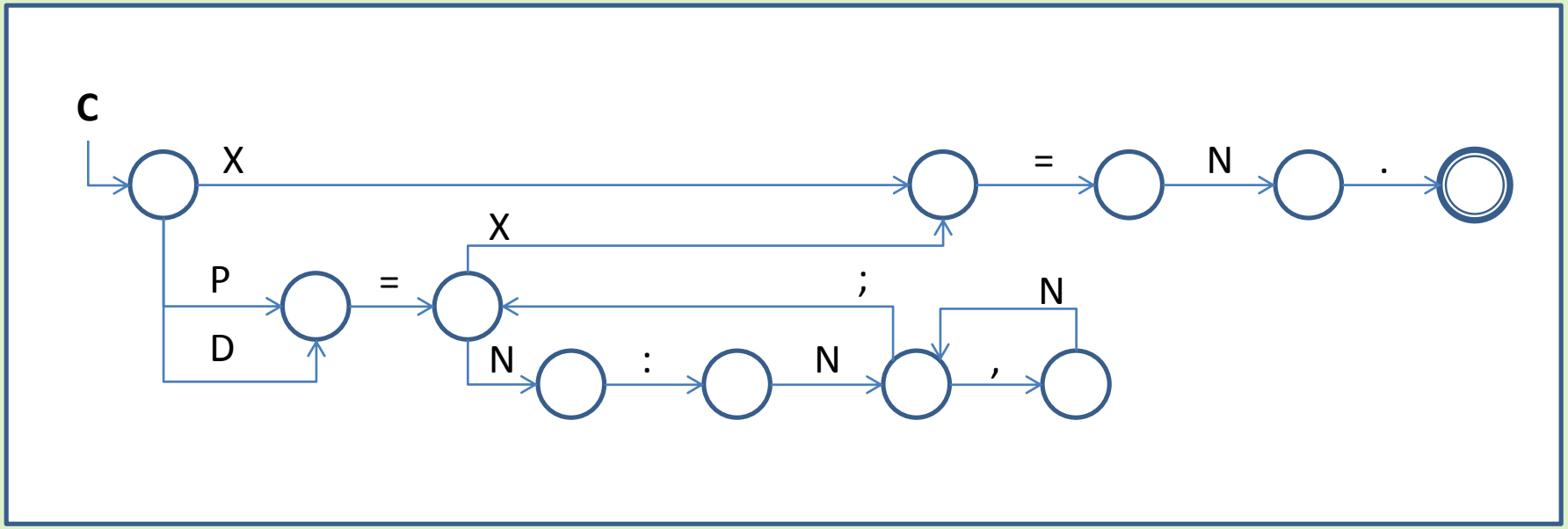
X = “X” “=” N “.” .

Reconhecedor resultante

- A partir desta gramática, obtém-se a expressão:

C = { (“P” | “D”) “=” { N “:” N { “,” N } “;” } } “X” “=” N “.” .

- E desta, resulta o autômato finito abaixo:



Formato do programa binário

- Um código C é formado por blocos de programa P, e/ou blocos de dados D, cada qual, carregado na memória a partir do endereço inicial correspondente. Ao final, indica-se o endereço de execução.
  (todos os numerais denotados em hexadecimal)

- Ex. 1: P = 100: 56, 45, 44, 23, 0F054;
  X = 100. (1 bloco apenas)

- Ex. 2: D = 02FF: 0;
  D = 12EA: 3, 3, 3, 3, 3;
  P = 0F80: 0DA, 1C, 9E;
  X = 0F80. (3 blocos de dados)

- Esta sintaxe é uma forma bastante simplificada, embora fiel, de formato de programa objeto binário executável (absoluto), podendo portanto ser utilizada na prática como formato de programas executáveis à imagem da memória.
- Um transdutor que seja capaz de ler essa sintaxe pode ser usado como núcleo de um carregador absoluto (loader) de um sistema de programação.

Operação típica

- Supondo-se que a memória esteja carregada com um programa em linguagem de máquina, que o seu contador de instruções esteja apontando a primeira instrução a executar, a operação típica da máquina de Von Neumann segue uma lógica similar à da calculadora do primeiro exemplo:

LOOP: Obter a próxima instrução a executar;

Executar o tratamento correspondente à instrução

Go to LOOP

Detalhamento da lógica de operação

- LOOP: buscar na memória a próxima instrução
- Decodificar a instrução obtida:
  - Se referência à memória,
    - Separar a operação especificada e os operandos
    - Obter na memória os valores associados aos operandos
    - Aplicar a operação especificada sobre os valores
- Executar a instrução
- Determinar a posição da próxima instrução
- Go to LOOP;

Semântica da operação

- Operation = On; {Load; Execute;} Off;
- Execute = { Fetch; Decode; Perform; }.

Fetch = RI ← Mem[CI]; CI←CI+1;

Decode = Extract (RI, Opcode, Operand)

Perform = case Opcode

...

i : i-th_instruction (Operand)

...

endcase

- Scheduler – determina o próximo evento a tratar.
- Decoder – interpreta o evento e escolhe para o mesmo a rotina de tratamento correspondente.
- Tratamento – executa o tratamento específico do evento que está sendo considerado.

Sintaxe da execução de um programa

- Load; Running← True; CI ← X; Run.
- Run = Wlhile Running do

```
{ Exec (Mem[CI]);
  if instrução-normal then CI++;
  if instrução-desvio then CI ← novoCI;
  if instrução-halt then running ← false.
 }
```

- O tratamento de cada instrução de máquina é definido pelo comportamento do hardware que implementa a arquitetura da máquina. Em geral, é feito em quatro etapas:
  - Busca na memória a próxima instrução a executar
  - Decodifica a instrução obtida, para determinar qual o procedimento que lhe corresponde
  - Promove a execução de tal procedimento
  - Atualiza o contador de instruções, apontando a próxima instrução a ser executada.

- No simulador, a decodificação e a execução da instrução costumam ser feitas por um comando de múltipla escolha, em função do código de operação da instrução, sendo executada apenas a opção associada à instrução.
- A lógica de simulação é portanto uma simples extensão da do primeiro exemplo, diferindo apenas
  - Quanto à fonte dos eventos (naquele caso, as teclas pressionadas, e neste, as instruções extraídas da memória)
  - Quanto ao sincronismo entre o simulador e os eventos (naquele caso, as teclas são pressionadas em momentos arbitrários, e neste, movas instruções são extraídas apenas após a simulação de cada instrução do programa).

Possível uso da adaptatividade

1. **Software adaptativo** – Além dos casos mencionados no primeiro exemplo, é possível empregar a adaptatividade na lógica do próprio programa a simular.

O código representa as operações do programa a executar, e a inclusão da adaptatividade o torna um código adaptativo, ou seja, auto-modificável.
A máquina abstrata de execução opera então como dispositivo subjacente de uma adaptatividade.

Há diversas implementações propostas para código adaptativo, como nas dissertações de Éder José Pellegrini e de Salvador Ramos Bernardino da Silva.

Uma implementação muito primitiva, aplicável a máquinas sem proteção ao código em hardware, consiste em simplesmente permitir que o código seja modificado dinamicamente pelo programa, sem que exceções sejam levantadas em reação a tal ocorrência.

Recentes avanços nesse sentido sugerem a utilização de sistemas de dispositivos adaptativos nos quais a camada adaptativa é implementada separadamente, de forma relativamente independente do dispositivo adaptativo que lhe é acoplado. Isso concede flexibilidade e transparência ao sistema, características muito desejáveis e tecnicamente convenientes.

2. Sistema simulado tem **funcionalidade dinâmica** – se a funcionalidade do sistema simulado puder ser modificada em funcionamento, sem a desconexão do sistema, ainda que temporária, isso caracteriza uma potencial adaptatividade.

Muitos sistemas modernos incorporam características dinâmicas, ainda que de forma primitiva, e um exemplo popular corresponde à reconfiguração dinâmica da maior parte dos sistemas operacionais que utilizem hardwares hospedeiros contendo **interfaces USB**, pois estas permitem o acoplamento/desacoplamento dinâmico de hardwares ao sistema, com a reconfiguração automática dos seus softwares de suporte.

O tratamento do evento relativo à reconfiguração exigida pode ser feito adaptativamente, por meio da instalação e ligação dinâmica ao sistema do código de software suplementar que se faça necessário, ou de sua desinstalação e desconexão, se for o caso.

Um formato mais usual

- Dependente de contexto:
- Uma sequência arbitrária de blocos da forma:

Nb Ei d1 d2 ... dNb Cs

onde

Nb = número de bytes >= 0

Ei = endereço inicial (do primeiro byte)

$d_1\ d_2 \ldots d_{Nb}$ = Nb dados (1 byte cada)

Cs = Checksum = soma dos dados mod 8

Especificação sintática

C = { B } .

B = L A { D } S ⊢ .

Dependências contextuais:

L = N -- -- -- (0 < val(N) < 128) *comprimento do bloco*

A = N -- -- -- (val(N) é um endereço válido) *endereço inicial*

D = N -- -- -- (-128 <= Val(N) <= 127) *dados*

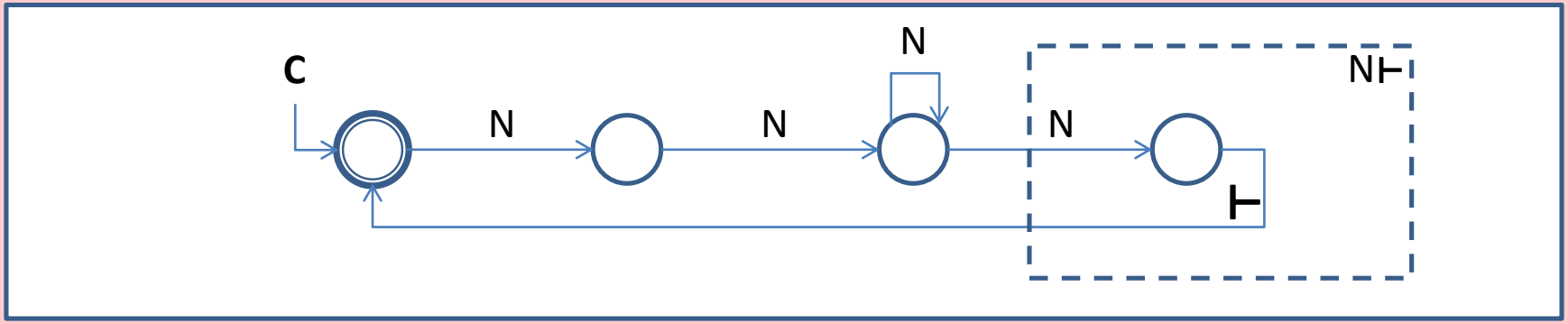
S = N -- -- -- (val(N) = (val(L) + val(A) + Σ (val(D))) mod 8)
checksum

Reconhecedor resultante

- Desta gramática, extrai-se trivialmente a expressão:

C = { N N { N } N ⊢} .

- E dela, é imediato o autômato finito abaixo.
(o não-determinismo no penúltimo estado é facilmente evitável se N ⊢ for pré-processado como token especial.)



Instanciação de uma simulação geral dirigida por eventos para representar um sistema operacional multiprogramado. Ênfase na representação dos eventos, do tempo, das simultaneidades, das corridas por recursos, das prioridades, das diferenças de tratamento devidas a condições de disponibilidade de recursos, de tempo, de prioridade etc. Ênfase na representação de concorrência e simulação de paralelismo em um hospedeiro sequencial.

APLICAÇÃO 3 – SIMULAÇÃO DE UM SISTEMA OPERACIONAL MULTIPROGRAMADO

Simuladores dirigidos por eventos

- A simulação de sistemas operacionais multiprogramados instancia uma simulação geral dirigida por eventos, com ênfase na modelagem de:
  - Concorrência
    - Competição
    - Colaboração
    - Prioridade
    - Compartilhamento
    - Paralelismo em hospedeiro sequencial.
  - Temporalidade
    - Tempo
    - Eventos
    - Simultaneidade
    - Sincronização
    - Comunicação

Modelagem da temporalidade: tempo e eventos

- Para modelar a temporalidade costumam ser criadas estruturas computacionais tais como:
  - Para a modelagem do **tempo**, uma variável designa o instante corrente de simulação, e uma lista ordenada pelo instante de ocorrência designa instantes para os quais estejam programadas ocorrências a serem simuladas. Uma característica desta variável é que é monotônica crescente, e seu valor nunca pode ser menor que o de uma outra variável global, o **instante corrente de simulação**.
  - Cada uma das ocorrências passíveis de simulação é dita um **Evento**, que é caracterizado pelo seu tipo e instante de ocorrência. Adicionalmente, outros atributos podem ser associados, como por exemplo, o programa a que se refere.

Lista de eventos

- Uma estrutura de dados fundamental em uma simulação dirigida por eventos é a **lista de eventos**, uma lista contendo todos os eventos programados conhecidos, que ainda não tenham sido tratados pelo sistema.
- Essa lista deve estar **ordenada** de acordo com o **instante de ocorrência previsto** para o evento em questão.
- A manipulação da lista de eventos prevê:
  - A retirada dos primeiros eventos da lista em primeiro lugar
  - Uso de critério de exceção em caso de anomalias temporais:
    - múltiplos eventos programados simultâneos.
    - primeiro evento foi programado para instante de simulação já passado.

Colisões e serialização

- Eventos programados para o mesmo instante podem ocorrer.
- Em computadores sequenciais é fisicamente impossível executar que dois programas ao mesmo tempo, por não haver paralelismo verdadeiro.
- Assim, seu tratamento deve ser serializado, em função de alguma regra arbitrária de desempate (por exemplo, por ordem alfabética, ou pelo instante de inserção na lista, ou qualquer outra).
- A **simultaneidade** do fenômeno real que estiver sendo simulado pode ser imitada pelo simulador através de técnicas de pseudo-paralelismo, de forma análoga ao que fazem os sistemas operacionais hospedados em processadores sequenciais, na implementação da multiprogramação.

Pseudo-paralelismo

- Mecanismos clássicos de **Sincronização** podem ser utilizados para garantir que processos pseudo-paralelos se mantenham sincronizados entre si e com o sistema. Para isso usam-se semáforos, regiões críticas e outras técnicas.
- Sistemas clássicos de **Comunicação** tornam-se cada vez mais importantes como recursos de implementação de pseudo-paralelismo, permitindo que processos se comuniquem entre si trocando mensagens e solicitando serviços um para o outro e ao sistema.

Concorrência

- Ambientes que executam processos concorrentes exigem recursos computacionais que permitam a execução de programas em regime de paralelismo verdadeiro ou de pseudo-paralelismo.
- Em tais situações, a limitação dos recursos disponíveis leva os processos a **competirem** pela sua posse, exigindo do sistema a adoção de políticas de alocação compatíveis com o ambiente operacional, com o perfil dos programas que o utilizam e com as metas adotadas para os índices de mérito do sistema.
- Em certos casos, um esquema de **prioridades** pode ajudar a desempatar impasses ocorridos nessas competições pela posse dos recursos (nesse caso – ver adiante – surge uma interessante oportunidade de utilização de técnicas adaptativas).

Compartilhamento

- Compartilhamento de recursos é uma técnica bastante antiga que se utiliza em programação paralela e concorrente, e que explora a possibilidade de permitir a mais de um processo o acesso controlado a um mesmo recurso compartilhável. Normalmente, o uso clássico de semáforos, regiões críticas e mensagens costuma ser explorado para a implementação do compartilhamento de recursos, como se pode encontrar em textos sobre sistemas operacionais, programação paralela e concorrente, redes de computadores etc.

Colaboração

- Processos costumam interagir em um ambiente através do uso compartilhado de recursos.
  Uma forma de compartilhamento é a **competição** pelo recurso, na qual o processo assume uma atitude predatória ao reivindicar para si a posse do recurso.
  Outra forma é a **colaboração**, e nesta o processo disponibiliza recursos para outros com os quais está colaborando, de forma que, através de um sistema de tomada de decisão, seja feita, dinamicamente, a escolha da atitude a ser tomada em relação a um dado recurso (por exemplo, retê-lo ou liberá-lo para uso por outro processo).
  Aqui surge outra oportunidade de aplicação da adaptatividade em sistemas desta natureza (ver adiante).

Paralelismo em hospedeiro sequencial

- O pseudo-paralelismo é simulado com naturalidade nossimuladores dirigidos por eventos, usando-se:
  - Uma lista ordenada de eventos (implementa a linha de tempo)
  - Uma política adequada de tratamento para casos de anomalias temporais (evento programado para o passado, ou mais de um evento programado para ocorrer em um mesmo instante).
- Isso permite que, com uma mínima serialização dos eventos simultâneos, e com a imposição de atrasos aos eventos não tratados ocorridos em instantes passados, seja adotada uma simulação serial aproximada apenas nesses casos especiais, preservando o rigor da simulação em todos os demais casos.
- Um bom planejamento das técnicas de simulação e, principalmente, do projeto da programação dos eventos, pode ajudar a reduzir drasticamente, até mesmo eliminando a ocorrência das anomalias temporais

Possível uso de adaptatividade

1) Quando processos competem pelo uso de recursos, é possível criar formas de resolução de impasses através da associação de prioridades aos processos, e de critérios de decisão de alocação que leve em conta essas prioridades. Em geral, adotam-se prioridades fixas, mas caso seja adotado algum algoritmo de decisão que permita **modificar dinamicamente as prioridades** dos processos em função de índices de mérito julgados oportunos, então técnicas adaptativas poderão ser **incorporadas ao processo de tomada de decisão** em questão, visando à otimização da operação do sistema.

Tomada de decisão: reter ou liberar

2) Na interação por **colaboração,** um mecanismo adaptativo de tomada de decisão pode ser incluído para, em função de índices de mérito ou algum outro critério adotado, **determinar dinamicamente se deve reter ou liberar um recurso compartilhável**, passando dessa forma o mecanismo adaptativo a fazer parte integrante das políticas de alocação do sistema operacional.

SUGESTÕES DE OUTRAS APLICAÇÕES

Ambientes visuais (simulação de universo), decomposição de cenas, granularidade de representação e simulação, multigranularidade, representação de cenas decompostas na forma de árvore adaptativa, organização e manutenção de informações das partes que compõem a cena, cada qual com sua granularidade, com o seu próprio nível de detalhe, e com seu próprio decaimento de memória temporal, visualizadores diversos para dar ao observador uma uniformidade de percepção da cena apesar das diferenças de granularidade e de decaimento temporal, uso da adaptatividade para representação interna apenas dos nós da árvore que realmente interessam, mecanismos adaptativos para permitir que as cenas se integrem coerentemente apesar das diferenças de granularidade e de detalhe e de decaimento temporal.

Muitas aplicações alternativas: simulação de universos, processador de macros, editoração de (hiper)textos estruturados, cursos, etc.

APLICAÇÃO 4 – SIMULAÇÃO DE UM AMBIENTE VISUAL, P/EX., PARA GAMES

Apresentação de sistemas de aquisição de dados: os dados adquiridos, os dados esperados, diagnósticos da comparação, apresentações de alarmes, reação a botões para acionamento de providências em malha aberta, atuação em malha aberta, sugestões de providências em malha aberta, escolha da providência a tomar, aplicação da providência escolhida,

APLICAÇÃO 5 – SIMULAÇÃO DE UM AMBIENTE DE SUPERVISÃO DE PROCESSOS

Apresentação de sistemas de aquisição de dados: os dados adquiridos, os dados esperados, diagnósticos da comparação, apresentações de alarmes, reação a botões para acionamento de providências em malha aberta, atuação em malha aberta, sugestões de providências em malha aberta, escolha da providência a tomar, aplicação da providência escolhida, mudança de modo de operação – malha aberta ou fechada -, decisões e atuações em malha fechada, críticas do operador a decisões tomadas em malha fechada

APLICAÇÃO 6 – SUPERVISÃO E CONTROLE DE PROCESSOS

Linguagens de alto nível, linguagens de propósito específico, simulação de máquinas reativas (ex.: máquinas de calcular, brinquedos, sistemas embarcados em geral), interfaces de operação para programas resultantes das saídas de meta-sistemas de propósito geral

APLICAÇÃO 7 – INTERPRETADORES

Interfaces homem-máquina para operação interativa, painéis de controle por botões, interfaces para operação de software de programação visual

APLICAÇÃO 8 – SISTEMAS REATIVOS

Tesaurus, coletor de nomes, front end para inferência sintática de línguas naturais, música, imagens, texto, etc.
Aplicação alternativa (em conjunto com expansor de macros [sintáticas]) para a criação de ambientes de editoração inteligentes, de ambientes para o desenvolvimento de programas etc.

APLICAÇÃO 9 – AQUISIÇÃO E ORGANIZAÇÃO DE DADOS SINTATICAMENTE ESTRUTURADOS

Evolução do anterior, inclui como meta o desenvolvimento de software, incluindo elementos do tipo semântico, dependente de contexto, referente a pré- e pós-requisitos para as expansões das macros [sintáticas] de tal forma que os textos resultantes não estejam coerentes apenas do ponto de vista léxico-sintático mas também em relação às dependências de contexto eventualmente presentes nas especificações.
Notações gráficas podem ajudar a desenvolver as interfaces do sistema com seu usuário, facilitando para este a representação das abstrações e das expansões.
O uso de linguagens de programação especialmente projetadas para servirem como linguagens de programação de software adaptativo torna mais próximo ainda a interação do programador com o ambiente.

APLICAÇÃO 10 – AMBIENTE PARA DESENVOLVIMENTO DE SOFTWARE

OBRIGADO.