Capítulo 3

Representação Seqüencial

> *... qu'aussitôt j'en fisse ma société. Ah! je vois que vous bronchez sur cet imparfait du subjonctif. J'avoue ma faiblesse pour ce mode, et pour le beau langage, en général. Faiblesse que je me reproche, croyez-le. Je sais bien que le goût du linge fin ne suppose pas forcément qu'on ait les pieds sales. N'empêche. Le style, comme la popeline, dissimule trop souvent de l'eczéma. Je m'en console en me disant qu'après tout, ceux qui bafouillent, non plus, ne sont pas purs.*
>
> *Tradução:*
>
> *... que de imediato eu a ele me associasse. Ah!, vejo que o senhor recrimina este imperfeito do subjuntivo. Confesso minha fraqueza por este modo, e pela bela linguagem em geral. Fraqueza que eu me condeno, creia-me. Sei bem que o gosto pelos tecidos finos não pressupõe obrigatoriamente que se tenha os pés sujos. Não obstante, o estilo, como a popeline, dissimula com muita freqüência o eczema. Eu me consolo dizendo que no fim das contas, aqueles que o desprezam também não são puros.*
>
> *Albert Camus,* La Chute.

Muitas vezes, da qualidade de uma forma de representação de um modelo depende a descoberta ou não da solução de um problema. Durante o projeto lógico seqüencial, múltiplas descrições de sistemas devem ser manipuladas. O objetivo deste Capítulo é discutir os modelos e as formas de representação seqüenciais de uso mais difundido no projeto lógico VLSI.

Com o intuito de suavizar a aridez característica desta parte da obra, inicia-se pela especificação de um estudo de caso de sistema seqüencial na Seção 3.1, que deverá guiar a introdução dos conceitos formais. Este estudo de caso é usado ao longo do Capítulo com a evolução do seu projeto entremeada às

definições necessárias.

O modelo formal básico para representar sistemas seqüenciais é o de *estrutura de transição de estados finita* (em inglês, *finite state transition structure* ou *FSTs*). A partir deste, derivam outros modelos importantes, tais como o de *máquinas de estado finitas* (em inglês, *finite state machines* ou FSM*s*) e *autômatos finitos* (em inglês, *finite automata* ou *FAs*), cada um com aplicações específicas. Autômatos finitos podem ser determinísticos (em inglês, *deterministic finite automata* ou *DFAs*) ou não-determinísticos (em inglês, *nondeterministic finite automata* ou *NFAs*); máquinas de estados finitas podem ser classificadas como máquinas de Mealy ou máquinas de Moore, de acordo com a dependência imediata ou não das saídas com relação às entradas, respectivamente. Apresentar tais modelos gerais é assunto da Seção 3.2.

Além dos modelos formais, explora-se na Seção 3.3 as formas de representação especificamente usadas para descrever sistemas síncronos. Entre as formas de maior uso, destacam-se o *grafo de transição de estados* (*state transition graph* ou STG) e a *tabela de transição de estados* (em inglês, *state transition table* ou STT).

Outras representações seqüenciais no nível lógico de abstração devem ser empregadas ao se desistir de trabalhar com a hipótese síncrona. A discussão de algumas destas formas é o tema da Seção 3.4. Os *grafos de transição de sinal* (do inglês, *signal transition graphs* ou ASTGs) são adequados para uma representação mais genérica de sistemas seqüenciais, junto com as *tabelas de fluxo* (do inglês, *flow tables* ou AFTs). Emprega-se o prefixo "A" nas abreviaturas para indicar a natureza assíncrona da forma de representação.

Coloca-se ênfase na análise comparativa destas formas quando empregadas para descrever sistemas síncronos e assíncronos. Por exemplo, é fundamental distinguir o conceito de *estado* em sistemas síncronos daquele em sistemas assíncronos. Deve-se também distinguir a estrutura da função de transição de estados em cada caso.

3.1 Um Estudo de Caso de Sistema Seqüencial

Nesta Seção, introduz-se um estudo de caso de projeto de um sistema digital seqüencial. Os passos necessários para ir desde uma especificação informal do problema até a implementação no nível lógico de abstração de diferentes soluções para o mesmo são apresentados nesta Seção e nas três Seções subseqüentes. Na seqüência apresentada aqui, introduz-se e discute-se informalmente conceitos relevantes de projeto, tais como *enunciado do problema*, *abstração*, *modelagem*, *restrições*, *corretude* e *otimalidade* de uma solução do problema.

No que concerne sistemas digitais seqüenciais, o estudo de caso servirá para introduzir e diferenciar as noções de sistemas digitais síncronos e assíncronos. As Seções seguintes apresentam modelos e formas de representação associadas

a implementações síncrona e assíncrona de um sistema digital que resolve o problema. Os modelos e formas de representação são apresentados formalmente e detalha-se instâncias dos modelos e das representações com base no estudo de caso, quando relevante. Aqui, apresenta-se o enunciado e a análise de requisitos do problema, concluindo com a escolha de uma estratégia geral abstrata de implementação.

Exemplo 3.1 (Controle de tráfego - enunciado) Considere a rodovia federal BR 2000 de alto tráfego, intersectada pela Picada do Mato, uma estrada vicinal freqüentada apenas eventualmente por algum automóvel ou implementos agrícolas. Se não existir controle de tráfego no cruzamento, é praticamente impossível que um veículo proveniente da Picada do Mato consiga atravessar a BR 2000 em segurança. Por outro lado, claramente faz pouco sentido instalar no cruzamento um semáforo que interrompa o tráfego na BR 2000 a cada 1 ou 2 minutos, pois em 99% das vezes não haveria veículos desejando cruzar a rodovia a partir da Picada. O enunciado do problema a resolver pode ser então: "Instalar um sistema de controle na intersecção da BR 2000 com a Picada do Mato de forma a:

1. garantir que um veículo querendo cruzar a BR 2000 a partir da Picada do Mato possa eventualmente fazê-lo;
2. garantir que um veículo cruzando a BR 2000 a partir da Picada de Mato faça-o com um alto nível de segurança;
3. não interromper inutilmente o fluxo de tráfego na BR 2000."

■

O enunciado contém uma série de informações úteis para se chegar à solução, mas não todas. Os itens enumerados no enunciado atuam como *restrições* a serem atendidas pela solução do problema. A partir do enunciado, pode-se iniciar a fase de *análise de requisitos*, onde se determina, um conjunto de estratégias para atingir a solução, e estuda-se a viabilidade de cada uma, gerando como resultado um subconjunto destas para fases posteriores.

Exemplo 3.2 (Controle de tráfego - análise de requisitos) Vejamos algumas estratégias possíveis para implementar o sistema de controle, com eventuais variações:

1. permitir o cruzamento pelo estabelecimento de prioridades reguladas por sinais de trânsito:
   (a) prioridade para veículos trafegando na BR 2000;

   (b) prioridade para veículos trafegando na Picado do Mato;
   (c) prioridade à direita.

2. instalar um semáforo:
   (a) com temporização fixa;
   (b) com controle de temporização ajustado manualmente;
   (c) com controle de temporização ajustado automaticamente.

3. eliminar o cruzamento:
   (a) construir um túnel sob a BR 2000;
   (b) construir um viaduto sobre a BR 2000.

A estratégia 1 é a de mais baixo custo de implementação. Contudo, a variação 1(a) não satisfaz o requisito 1, e os veículos da Picada do Mato podem ter de esperar indefinidamente para cruzar a BR. A variação 1(b) atende a todos os requisitos, mas não constitui uma solução razoável. Suponha que durante o período da colheita, o fluxo de veículos na Picada do Mato aumente consideravelmente, truncando de forma inaceitável o tráfego da BR 2000. Em verdade, esta análise indica a ausência no enunciado de um requisito contemplando a importância relativa de garantir o fluxo em cada uma das vias. Em geral, especificações podem pecar por deixar implícitas situações como esta. Sabemos que garantir o fluxo na BR 2000 deveria ser mais relevante. Como conclusão, deve-se ter em mente que revisões do enunciado podem ser necessárias durante a fase de análise dos requisitos. A variação 1(c) é a de mais baixo custo, pois a legislação de trânsito do País assume prioridade à direita na falta de sinalização específica. Entretanto, e isto vale para todas as variações desta estratégia, a experiência com trânsito leva a concluir que a segurança desta variação é baixa, ainda que tecnicamente resolva o problema. Ou seja, o requisito 2 não está plenamente atendido.

A estratégia 2 aumenta consideravelmente a segurança mas possui custo maior. A variação 2(a) não atende o requisito 3, o mesmo ocorrendo com a variação 2(b). Esta última permite ajustes, minimizando os custos da variação 1(a). No limite, esta variação pode ser tornada quase perfeita, pela colocação de um funcionário controlando visualmente o cruzamento, mas a um custo hoje considerado proibitivo. A variação 2(c) aumenta o custo com relação à 2(a), mas apenas de forma marginal, dado o baixo preço atual de sistemas de controle automáticos, sobretudo os baseados em sistemas digitais. O controle automatizado, baseado no sensoriamento de passagem de veículos nas vias, e na atuação sobre um semáforo, pode ser a melhor estratégia para atender exatamente todos os requisitos do problema, e será a variação escolhida para o desenvolvimento do resto deste estudo.

Embora 3 seja sem dúvida a estratégia mais segura, o custo elevado pode descartar todas suas variações. Embora seja válido argumentar que não se deve privilegiar baixo custo em detrimento de segurança, construir um túnel ou viaduto pode ser tão caro a ponto de ultrapassar o orçamento disponível, e esta estratégia assim é descartada por inviabilidade, não por compromisso.

Note-se que as estratégias e variações arroladas aqui não são as únicas (poder-se-ia considerar por exemplo construir um túnel sob a Picada do Mato, entre outras soluções), mas durante a análise de requisitos, a experiência do profissional encarregado do problema auxilia-o a descartar automaticamente a grande quantidade de soluções que não apresentam uma relação custo-benefício razoável.

■

As Seções a seguir discutem outros aspectos da solução do Exemplo, cada uma abordando uma fase distinta da modelagem e solução do problema.

3.2 Modelos Seqüenciais

O ponto de partida para a proposta de modelos de sistemas seqüenciais são as definições de sistema digital (Página 2), de estado de um sistema digital (Página 11) e de sistemas digitais combinacionais e seqüenciais (Página 12). Definir estes conceitos de forma geral é uma tarefa difícil, e optou-se no Capítulo 1 por definições estruturais, que tornam mais fácil sua interpretação. Aqui contudo, alguns destes termos vão receber uma visão alternativa, relacionada ao comportamento do sistema subjacente. Tal visão é também intuitiva e mais adequada à manipulação de sistemas seqüenciais descritos abstratamente.

Primeiro, observe-se a diferença entre sistemas combinacionais e seqüenciais. Um sistema digital combinacional pode ser definido, abstraindo suas características de atraso de propagação (t_{pd}) do efeito de mudanças nas entradas para as saídas, como: *todo e qualquer sistema cujo comportamento das saídas depende única e exclusivamente do valor instantâneo das entradas.* Ou seja, para cada combinação distinta de valores de entradas, se estas são mantidas estáveis por um tempo suficientemente longo[1], as saídas apresentam valores determinados apenas por esta combinação. Por oposição, define-se sistema digital seqüencial como: *todo e qualquer sistema cujo comportamento das saídas não depende única e exclusivamente do valor instantâneo das entradas, mas também da seqüência de entradas recebidas anteriormente.* Ou seja, sistemas seqüenciais *armazenam* fatos do passado de entradas, e reagem de acordo com este conhecimento do passado e do valor das entradas atuais.

[1] A título de exemplo, hoje, em sistemas digitais VLSI do estado da arte, os módulos combinacionais mais complexos, tais como por exemplo multiplicadores de 32 ou 64 bits, apresentam valores de t_{pd} tipicamente abaixo do milionésimo de segundo ($10^{-6}s$).

Surge desta distinção de tipos de sistemas uma nova noção, comportamental, de estado. Estado é então: *a informação armazenada internamente no sistema digital sobre o passado das entradas.* Note-se que esta nova definição é totalmente compatível com a da Página 11, pois sistemas combinacionais podem ser vistos sob esta nova óptica como contendo exatamente *um* estado. Eles não podem *trocar* de estado, logo se *lembram* sempre da mesma informação. Como os teóricos demonstram, uma informação imutável não pode ser qualificada como tal [165]. Logo, sistemas combinacionais não se *lembram* de seu passado. Sistemas seqüenciais possuem pelo menos dois estados, e são capazes de armazenar tanta informação quanto permita sua cardinalidade de estados[2].

Outra idéia importante, que deriva da discussão acima é a de que um sistema digital está a cada momento em um dado estado (sempre o mesmo para sistemas combinacionais) e as entradas fornecidas ao sistema geram mudanças, que se denominam *transições de estado.*

Enquanto aparatos construídos fisicamente, sistemas digitais têm necessariamente um número *finito* de estados, pois o armazenamento de uma quantidade infinita destes implicaria o uso de uma quantidade infinita de recursos de hardware. Portanto, um modelo de sistemas seqüenciais prático não precisa ser capaz de representar sistemas com mais do que uma quantidade fixa, finita de estados. Isto não signfica que tais modelos não sejam úteis em projeto VLSI. Contudo, restringir-se a modelos finitos facilita o raciocínio e a manipulação formal.

Cabe salientar que somente variações nas entradas podem gerar transições de estado no sistema. Logo, o sistema digital gera saídas em resposta a entradas, e ao mesmo tempo muda de estado. Aceitando-se que informação e tempo são discretizados, tem-se um modelo onde uma seqüência de comprimento n de entradas corresponde a uma seqüência de comprimento n de saídas, ou, se a última entrada ainda não foi processada, a uma seqüência de comprimento $n-1$ de saídas. Um cuidado a ser tomado nesta modelagem é que este raciocínio pressupõe que cada entrada permaneça estável tempo suficiente para gerar a saída correspondente, caso contrário ou ela é irrelevante, ou o sistema pode não funcionar de acordo com o previsto. Cada combinação de entradas ou de saídas recebe a denominação geral de *símbolo.*

Os diferentes modelos a apresentar são quatro, denominados genericamente de ***sistemas de transição de estados finitos***. Eles são intuitivamente definidos abaixo [90]:

- FSTs (Estrutura de transição de estados finita, em inglês *finite state transition structure*) - quando em um dado estado, recebem um símbolo e realizam uma transição para um novo estado;

[2] Daí deriva, por exemplo, o poder computacional dos microprocessadores modernos, capazes de possuir uma cardinalidade *bruta* de estados na faixa de 10^{1000}.

- FAs (Autômatos Finitos, em inglês *finite automata*) - são FSTs que percebem e eventualmente anunciam o atingimento de estados especiais, ditos *de aceitação*;
- RLs (Linguagens regulares, em inglês *regular languages*) - são conjuntos de seqüências de símbolos de entrada e saída (em inglês, *strings*). São o tipo de conjuntos de seqüências aceitas por FAs ou geradas por FSMs (definidas no próximo item). O nome regular vem de sua estrutura subjacente;
- FSMs (Máquinas de estados finitas, em inglês *finite state machine*) - são FSTs que geram ou emitem um símbolo de saída ao mesmo tempo que recebem uma entrada e executam uma transição de estado;

Os modelos intuitivamente definidos acima são ilustrados na Figura 3.1, onde exemplifica-se com FAs que cada um dos modelos pode ser subdividido em uma versão determinística (sigla correspondente precedida por D) e uma versão não-determinística (sigla precedida por N).

3.2.1 Modelos Para o Comportamento Interno

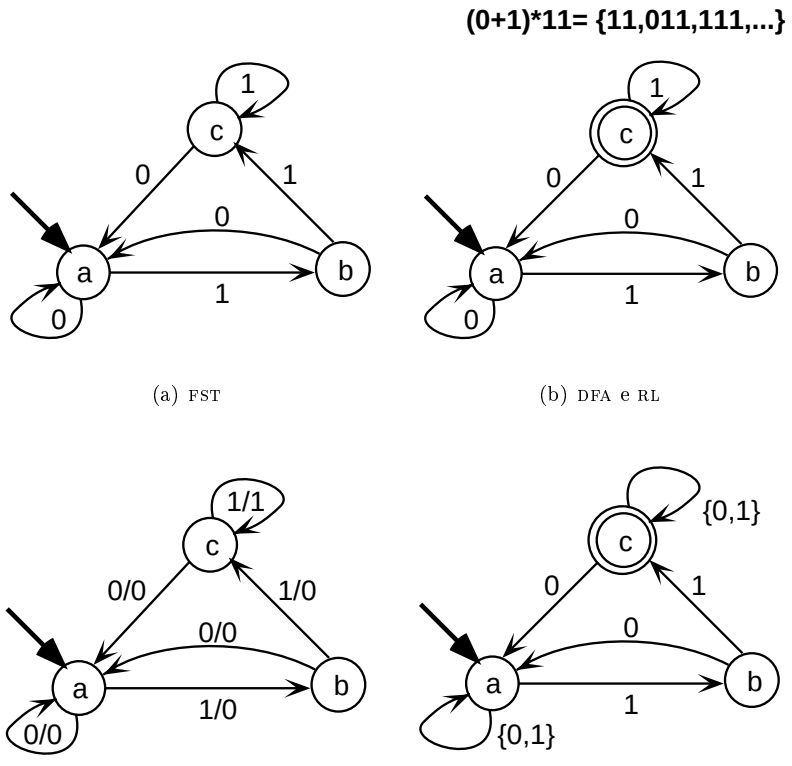
Descreve-se agora formalmente os modelos para representar o comportamento seqüencial de sistemas digitais, iniciando com modelos para representar o comportamento interno de transição de estados. Mais tarde, adiciona-se funcionalidade a estes para produzir modelos capazes de representar ambos, o comportamento interno e a produção de saídas.

Definição 3.1 (Estrutura de transição de estados finita) *Uma* ***estrutura de transição de estados finita*** *(*FST*) é uma estrutura algébrica* $\mathcal{T} = \langle I, S, \delta, S_0 \rangle$ *onde:*

1. $I = \{i_{p-1}, i_{p-2}, \ldots, i_0\}$ *é o* ***alfabeto de entradas****;*
2. $S = \{s_{q-1}, s_{q-2}, \ldots, s_0\}$ *é o* ***conjunto*** *ou* ***alfabeto de estados****;*
3. δ *é a função discreta* $\delta : I \times S \longrightarrow \mathcal{P}(S)$*, chamada* ***função de transição*** *ou* ***função próximo estado*** *de* $\mathcal{T}$*;*
4. $S_0 \subseteq S$ *é um* ***conjunto de estados iniciais***FST***!conjunto de estados iniciais****.*

O valor $\delta(i_j, s_k)$ *é denominado uma* ***transição*** *de* $\mathcal{T}$*.*

Uma FST $\mathcal{T}$ *é dita* ***determinística*** *(*DFST*) se a imagem de todos os pares* (i, s) *for um conjunto unitário. Se* S_0 *também é um conjunto unitário,* $\mathcal{T}$ *é dita* ***fortemente determinística****. Se o mapeamento* δ *for especificado para todo par* $(i, s) \in I \times S$*,* $\mathcal{T}$ *é uma* FST ***completa****. Uma não-especificação é equivalente à especificação* $\delta(i, s) = S$*.*



(a) FST (b) DFA e RL

(c) FSM (d) NFA

Figura 3.1: Modelos para sistemas de transição de estados finitos. Vértices representam estados, arestas representam transições, e rótulos nas arestas representam ou entradas ou pares entrada/saída, no caso da FSM (c) ou conjuntos de entradas associadas à transição (d). Setas sem vértice de origem indicam estados iniciais. Círculos duplos são estados de aceitação. Em (b), mostra-se também a linguagem regular aceita pelo DFA, no caso, qualquer seqüência de caracteres do conjunto $\{0,1\}$ que termina por um par de 1s.

Dada a definição de FST, alguns conceitos adicionais são necessários antes de apresentar os modelos restantes.

Definição 3.2 (Produto de FSTs) *Dadas n FSTs $\mathcal{T}_j = \langle I, S_j, \delta_j, S_j^0 \rangle$, com $1 \leq j \leq n$, define-se o* ***produto*** *de FSTs, $\Pi_j M_j$ como a FST*

$$M \quad = \quad \langle X, S, \delta(x,s), S^0 \rangle, \quad \textit{onde}$$

$$\begin{aligned} S &= \overset{j=n}{\underset{j=1}{\times}} S_j, \\ S^0 &= \overset{j=n}{\underset{j=1}{\times}} S_j^0, \\ \delta(x,(s_1,\ldots,s_n)) &= \overset{j=n}{\underset{j=1}{\prod}} \delta_j(x,s_j) X \times S \longrightarrow \mathcal{P}(S). \end{aligned}$$

Assume-se que todos os S_j são disjuntos. Uma FST *é determinística se sua função de transição $\delta_j(x,s_j)$ é. Se cada* FST *acima é determinística, o produto também é.*

Definições 3.3 (Execução, cadeia) *Uma* ***execução*** *ou* ***run*** *de um* FST *é um conjunto finito ou não, totalmente ordenado de estados onde o primeiro destes é algum estado inicial $s_0 \in S$ e o conjunto ocorre em resposta a algum conjunto ordenado de entradas possível. O conjunto de entradas pode ser um string (Definição 2.33, Página 61) ou uma fita (Definição 2.35), caso seja finito ou infinito, respectivamente. A combinação string ou fita e uma execução é chamada* ***cadeia****.*

Definição 3.4 (Autômato finito) *Um* ***autômato finito*** *(*FA*) é uma* FST *com um* ***conjunto aceitação*** *$A \subset S$ ou seja uma estrutura algébrica $\mathcal{A} = \langle \mathcal{T}, A\rangle = \langle I, S, \delta, S_0, A\rangle$ onde A define o conjunto de* ***estados finais*** *ou* ***estados de aceitação****, determinados como segue. Considere-se um string de entrada $\mathbf{i} = (i_1,\ldots,i_n)$, com $i_k \in I, \forall k, 0 \leq k \leq n$. Dado um estado inicial $s_0 \in S_0$, $\mathbf{i}$ produz uma execução $\mathbf{s} = s_0^i,\ldots,s_n^i$, onde $s_k^i \in \delta(s_{k-1}^i, i_k) \subseteq S$. O string $\mathbf{x}$ é* ***aceito*** S.S.S. *$s_n^i \in A$.*

O autômato FA *é* ***determinístico*** *se a função de transição δ do* FST *subjacente o é, e é* ***completo*** *se δ for completa.*

Antes de definir linguagem regular, é preciso abordar o conceito formal de linguagem em si. A partir do conceito de aceitação de strings em FAs, diz-se que o conjunto de strings aceitos por um FA é uma linguagem[3]. Esta é a ***linguagem*** do FA. Obviamente, se o conjunto aceitação do FA é o conjunto vazio, nenhum string é aceito pelo FA, e a linguagem correspondente é vazia.

Definição 3.5 (Linguagem formal) *Uma* ***linguagem formal*** *(*FL*) é um conjunto de seqüências finitas ou infinitas, ou seja, de strings ou fitas, respectivamente, onde admite-se a palavra vazia ϵ como elemento da linguagem. Denota-se uma linguagem por $\mathcal{L}$, eventualmente usando subscritos. $\mathcal{L} = \emptyset$ é a linguagem vazia, e $\mathcal{L} = \{\epsilon\}$ é a linguagem unitária contendo apenas a palavra vazia. Note-se então que $\{\epsilon\} \neq \emptyset$. Usando o operador estrela (definido*

[3]De fato, esta é uma definição geral para linguagens regulares, equivalente à definição formal apresentada aqui

na Página 62), estabelece-se uma relação entre estes conjuntos, pois por convenção, $\emptyset^* = \{\epsilon\}$. O **produto** *de duas linguagens $\mathcal{L}_1, \mathcal{L}_2$ é definido como o produto Cartesiano dos conjuntos:*

$$\mathcal{L}_1\mathcal{L}_2 = \{\mathbf{x}_1\mathbf{x}_2 \mid \mathbf{x}_1 \in \mathcal{L}_1, \mathbf{x}_2 \in \mathcal{L}_2\}.$$

Como linguagens são conjuntos, está igualmente bem definida a **união** *de linguagens, denotada de maneira convencional ou usando a notação aditiva:*

$$\mathcal{L}_1 \cup \mathcal{L}_2 = \mathcal{L}_1 + \mathcal{L}_2 = \{\mathbf{x} \mid \mathbf{x} \in \mathcal{L}_1 \textit{ ou } \mathbf{x} \in \mathcal{L}_2\}.$$

Definição 3.6 (Linguagem regular) *Considere-se dado um alfabeto X. X^* é o conjunto infinito de todos os strings finitos sobre X, incluindo a palavra vazia. X^* é obviamente uma linguagem formal, sendo denotado alternativamente por $\mathcal{L}(X)^*$ para reforçar esta identidade. Uma linguagem (ou um conjunto) $\mathcal{X} \subseteq \mathcal{L}(X)^*$ é uma* ***linguagem regular*** *(RL), também denominada* ***conjunto regular****, notada $\mathcal{X} \in \mathcal{R}(X)$, se ela pode ser obtida a partir do conjunto vazio e dos conjuntos unitários $\{x\} \subseteq X$ por um número finito de operações união, produto e estrela.*

Vale a pena mostrar exemplos de linguagens regulares. Uma forma compacta de representar estas linguagens é usando ***expressões regulares***. Esta forma emprega os elementos do conjunto de símbolos X, a representação da palavra vazia (ϵ), o conjunto vazio ($\emptyset$), combinados por operações de união ($+$), produto (implícita na colocação lado a lado de elementos), e os expoentes $+$ e $*$ (operação estrela). Parênteses são usados para mudar a precedência implícita, onde operações produto e estrela tem precedência sobre a operação união. A Tabela 3.1 mostra exemplos de expressões e linguagens regulares correspondentes assumindo o conjunto $X = \{0, 1\}$.

Concluindo esta Seção, coloca-se aqui, sem prova, o principal resultado da teoria de linguagens regulares.

Teorema 3.1 (Equivalência entre FAs e RLs) *Para toda linguagem regular $\mathcal{L}$, existe um autômato finito determinístico (DFA) $\mathcal{A}(\mathcal{L})$, cuja linguagem é $\mathcal{L}$. Por outro lado, a linguagem $\mathcal{L}(\mathcal{A})$ de qualquer DFA é regular.*

O teorema acima demonstra a total equivalência dos modelos DFA e RL. O leitor interessado pode recorrer a qualquer boa obra da área de linguagens formais para a demonstração do Teorema 3.1. Exemplo é o livro de Cohen, [52]. A prova é construtiva, fornecendo algoritmos recursivos de conversão de FAs em RLs e vice-versa.

Tabela 3.1: Exemplos de expressões e linguagens regulares.

Expressão regular	Linguagem regular	Descrição
ϵ	$\{\epsilon\}$	Conjunto com um elemento, a palavra vazia.
$\emptyset$	$\{\}$	A linguagem vazia.
$(00+11)$	$\{00, 11\}$	Conjunto com dois elementos, 00 e 11.
$0^+(00+11)$	$\{000, 011, 0000, 0011, \ldots\}$	Conjunto infinito, linguagem de todos os strings que possuem um número arbitrário maior do que 1 e finito de zeros, e que termina por 00 ou por 11.
$(0+1)^*001$	$\{001, 0001, 1001, \ldots\}$	Conjunto de strings que terminam por 001.

3.2.2 Modelo Geral

Dadas as definições de modelos para representar o comportamento interno de transição de estados de um sistema seqüencial, pode-se agora generalizar, acrescentando capacidade de gerar saídas a estes. O novo modelo assim obtido é a máquina de estados finita.

Definição 3.7 (Máquina de estados finita) *Uma* ***máquina de estados finita*** *(*FSM*) é uma sêxtupla* $\mathcal{M} = \langle I, S, O, \delta, \lambda, S_0 \rangle$ *onde:*

1. $I = \{i_{p-1}, i_{p-2}, \ldots, i_0\}$ *é o* ***alfabeto de entradas****;*
2. $S = \{s_{q-1}, s_{q-2}, \ldots, s_0\}$ *é o* ***conjunto*** *ou* ***alfabeto de estados****;*
3. $O = \{o_{r-1}, o_{r-2}, \ldots, o_0\}$ *é o* ***alfabeto de saídas****;*
4. δ *é a função discreta* $\delta : I \times S \longrightarrow S$*, chamada* ***função próximo estado*** *de* $\mathcal{M}$*; dado um par* $(i_j, s_k) \in I \times S$*, se* $\delta(i_j, s_k)$ *está especificada,* $s_l = \delta(i_j, s_k)$ *é o* ***próximo estado*** *da* FSM $\mathcal{M}$ *que corresponde à entrada* i_j *e ao* ***estado atual*** s_k*;*
5. λ *é uma função discreta* $\lambda : I \times S \longrightarrow O$*, chamada* ***função de saída*** *de* $\mathcal{M}$*; dado o par* $(i_j, s_k) \in I \times S$*, se* $\lambda(i_j, s_k)$ *está especificada,* $o_m = \lambda(i_j, s_k)$ *é a* ***saída*** *da* FSM $\mathcal{M}$ *que corresponde à entrada* i_j *e ao estado atual* s_k*;*
6. $S_0 \subseteq S$ *é um* ***conjunto de estados iniciais****.*

O par $(\delta(i_j, s_k), \lambda(i_j, s_k))$ *é chamado uma* ***transição*** *da máquina* $\mathcal{M}$.

Quando ambas, δ *e* λ *são funções completas, a* FSM *é* ***completa*** *ou* ***completamente especificada*** *(*CFSM*); senão, ela é* ***parcial*** *ou* ***incompletamente especificada*** *(*ISFSM*).*

A definição presente é uma ***máquina de Mealy*** [134]. *Se a função de saída não depender da entrada, ou seja, se* $\lambda : S \longrightarrow S$, *trata-se de uma* ***máquina de Moore*** [138].

Em situações em que o conjunto de estados iniciais é irrelevante, pode-se representar uma FSM *mais compactamente como uma quíntupla* $\mathcal{M} = \langle I, S, O, \delta, \lambda \rangle$.

A primeira observação que se pode fazer sobre a definição de FSM é que ela poderia ser mais geral, seja definindo δ e λ da mesma maneira que foi definida δ para FSTs, com a imagem da função sendo o conjunto potência de S, ou equivalentemente definindo δ e λ como relações binárias. Este último caso geral é tratado extensamente por Zahnd em [197]. Restringindo δ e λ a serem funções sobre o domínio S, limita-se o modelo, que é então capaz de descrever apenas *máquinas seqüenciais do tipo padrão* (do francês *machines séquentielles du type standard*), de acordo com a terminologia introduzida em [197]. Entretanto, aqui interessa manipular somente máquinas seqüenciais que apresentem um comportamento *determinístico* (como definido por Kohavi em [115]), as únicas definitivamente implementáveis sob a forma de hardware. A definição provida aqui é capaz de representar qualquer máquina determinística [115] e mesmo uma forma limitada de não-determinismo, habilitada pelo uso de funções incompletamente especificadas.

Se as transições especificadas de uma FSM nunca incluírem como elemento do codomínio a palavra vazia, os dois tipos de máquina, Mealy e Moore, são equivalentes. Ou seja, qualquer comportamento descrito usando uma destas pode ser descrito usando a outra e vice versa, conforme demonstra Kohavi em [115]. A Definição 3.7 implica comportamentos que jamais incluem a palavra vazia, devido mais uma vez à forma da função δ, e é coerente com a conceito inicial de seqüências (Definição 2.33).

Continuando o exemplo introduzido na Seção 3.1, vejamos como o conceito de FSM pode ser usado para modelar o sistema digital que resolve o problema do cruzamento da BR 2000 com a Picada do Mato. O grau de abstração da presente modelagem será alto, deixando discussões detalhadas para as Seções seguintes, onde exploram-se as alternativas síncrona e assíncrona para implementação do sistema.

Exemplo 3.3 (Controle de tráfego - modelagem abstrata) Como foi decidido no Exemplo 3.2 onde se realizou a análise de requisitos, quer-se instalar um semáforo no cruzamento da BR 2000 com a Picada do Mato, com um controle de temporização ajustado automaticamente. Continua-se agora o trabalho, estabelecendo a forma geral do sistema digital de controle.

Supondo que ambas a vias sejam transitáveis em ambos sentidos, existe a necessidade de instalar 4 semáforos, uma para cada sentido de fluxo de tráfego em cada uma das vias. Para simplificar a discussão apenas, ignora-se a existência da luz de atenção (o amarelo) nos semáforos. Claramente, na prática esta não é uma abstração interessante, sequer viável de ser feita.

Em primeiro lugar, deve-se estudar a estrutura da interface entre o sistema digital de controle dos semáforos e o mundo externo. O sistema de sensoriamento das vias poderia captar quatro sinais indicando presença ou não de veículo sobre cada um dos sentidos de cada via. Contudo, como a intenção é que o semáforo mude apenas em resposta à presença de veículos na Picada, apenas sensores sobre a Picada fazem sentido. Além do mais, é indiferente qual dos lados da Picada é considerado, apenas importando para o sistema a existência ou não de algum veículo nesta. Assim, antes da entrada do sistema de controle, os sinais dos dois sensores podem ser combinados em uma única informação binária, indicando a existência ou não de veículo na Picada. Denomine-se este sinal de VP (abreviando "Veículo na Picada"), e codifique-se $V = 1$ se existe veículo na Picada e $VP = 0$, caso contrário.

Além disso, existe uma discrepância clara entre a velocidade de um sistema digital e a temporização de um semáforo. Logo, é necessário fornecer como entrada para o sistema uma referência temporal adequada. Supõe-se aqui uma entrada binária periódica denominada RT (abreviando "Referência Temporal"), que permanece 70s em 0 e 30s em 1.

Embora existam 256 combinações possíveis de luzes acesas ou apagadas nos semáforos (existem 8 luzes, cada uma pode estar acesa ou apagada), apenas duas combinações distintas são *válidas*, todas as outras correspondendo a situações indesejáveis[4]. As combinações em questão são:

- luzes vermelhas acesas nos dois semáforos da Picada e luzes verdes acesas nos dois semáforos da BR, todas as luzes restantes apagadas;
- luzes verdes acesas nos dois semáforos da Picada e luzes vermelhas acesas nos dois semáforos da BR, todas as luzes restantes apagadas.

Assim, o sistema digital em questão pode ser pensado como possuindo uma saída binária, indicando a combinação a ser ativada a cada instante. Denomine-se esta saída VB (abreviando "Vermelho na BR"), e codifique-se $VB = 1$ se os semáforos da BR 2000 devem estar fechados e os da Picada abertos, e $VB = 0$ para a situação oposta. A Figura 3.2 mostra o diagrama de blocos do sistema de controle a ser implementado, com as entradas e saídas discutidas acima.

[4]Este comportamento indesejável pode ser usado para especificar e construir *sistemas tolerantes a falhas*, bem como para estabelecer procedimentos de teste e simulação de falhas para o sistema final.

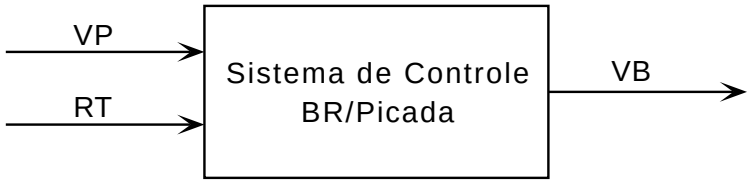


Figura 3.2: Diagrama de blocos do sistema de controle de tráfego a ser implementado no cruzamento da BR 2000 com a Picada do Mato. Cada uma das entradas e a saída são fios que transportam um valor binário.

Ataca-se agora a modelagem do comportamento do sistema digital em questão. Assume-se que a referência temporal designa intervalos em que o semáforo está aberto para a BR 2000 (os 70s durante os quais $RT = 0$) e em que o semáforo pode estar aberto para a Picada (os 30s durante os quais $RT = 1$). Esta organização permite garantir que nenhuma das vias irá sofrer postergação indefinida para usar o cruzamento. Com isto pode-se modelar a estrutura de transição de estados do sistema, de forma ainda bastante abstrata, como uma FSM de dois estados. No primeiro, batizado de a e que convencionase ser o estado inicial, o semáforo fica aberto para a BR (o que ocorre com mais freqüência), no segundo, batizado de b, o semáforo fica aberto para a Picada. Ao ocorrer uma transição de 0 para 1 o sinal RT, amostra-se os valores do sensores através do sinal VP. Caso o estado atual seja o que estabelece semáforo aberto para a BR, e caso $VP = 1$, há veículo aguardando na Picada e deve haver um transição de estado, e conseqüentemente da situação dos semáforos. Neste novo estado, uma transição de 1 para 0 do sinal RT, independente do valor do sinal VP, provoca o retorno ao estado inicial.

A Figura 3.3 mostra graficamente o comportamento sugerido do sistema digital como uma FSM.

Note-se que se trata de uma máquina de Moore, pois a saída depende apenas do estado atual da máquina, por isto as saídas estão representadas junto aos estados, e não junto às transições. Note-se também que enquanto o nível lógico da entrada VP é relevante, apenas as transições de valor do sinal RT são consideradas.

Neste ponto, a solução do problema foi modelada abstratamente como uma FSM. Os próximos passos são estudar os compromissos de implementação e decidir por alguma estratégia de construção do sistema digital. Um passo anterior à implementação deveria ser a validação da FSM aqui obtida, seja via simulação de seu comportamento, seja pela aplicação de técnicas de verificação formal. O presente exemplo é simples demais para justificar o emprego destes métodos, e o passo de validação da especificação abstrata é evitado aqui.

■

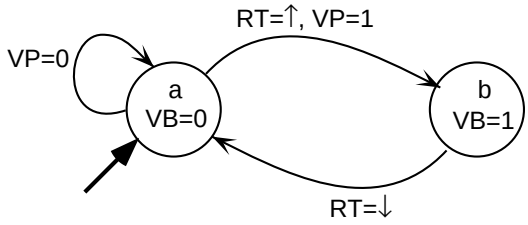


Figura 3.3: FSM para descrever o comportamento do sistema de controle de tráfego a ser implementado no cruzamento da BR 2000 com a Picada do Mato.

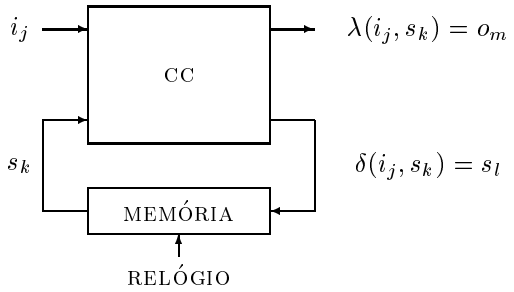
3.3 Representações Síncronas

Uma FSM pode ser representada de várias maneiras. Duas formas explícitas de representação (por oposição às formas implícitas introduzidas na Seção 2.2.3) destacam-se como de maior emprego, as baseadas em grafos e as baseadas em tabelas. Para sistemas digitais síncronos, estas representações correspondem respectivamente ao grafo de transição de estados, STG, e a tabela de transição de estados, STT. STGs já vêm sendo usados informalmente ao longo deste Capítulo.

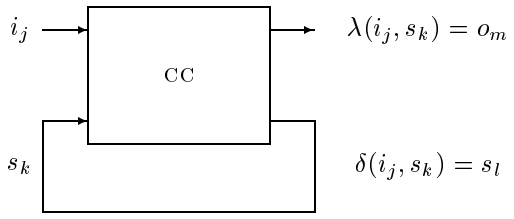
Uma máquina de estados finita descreve um processo iterativo; como tal, ela pode ser implementada de diferentes maneiras em hardware:

1. como uma ***implementação combinacional***, pela conexão *em cascata* [62] de células, realizando as funções próximo estado e de saída;
2. como uma ***implementação seqüencial síncrona***, pela conexão em laço fechado de um circuito combinacional (CC) que realize as funções próximo estado e de saída, e uma memória controlada pelo sinal de relógio;
3. como uma ***implementação seqüencial assíncrona***, similar à anterior, mas onde elementos combinacionais e seqüenciais estão espalhados na estrutura da implementação, sem nenhum sinal de relógio global para controlar o comportamento de transição de estados;
4. como uma ***implementação mista***, que surge de um estudo de compromisso de implementações "puras", e que usualmente resulta em uma rede hierárquica ou não de blocos de hardware que se comunicam.

Nesta Seção, refere-se à implementação de uma FSM mediante a estratégia seqüencial síncrona. Tal implementação é mostrada esquematicamente na Figura 3.4(a) para uma máquina de Mealy. A Figura 3.4(b) mostra o modelo de implementação assíncrono, a ser tratado na Seção 3.4. No que segue, CC representa a ***parte combinacional*** da FSM.



(a) Modelo síncrono



(b) Modelo assíncrono

Figura 3.4: Representação esquemática de máquinas de estados finitas síncrona (a) e assíncrona (b) como uma máquina de Mealy. CC é um sistema digital combinacional responsável por implementar as funções próximo estado e de saída. Entradas, saídas e estados são representados na Figura de maneira simbólica. (a) A memória é controlada pelo sinal de relógio, que comanda a realimentação. (b) Não existe sinal de relógio, o comportamento seqüencial é dado pelos laços de realimentação, que pressupõe um atraso entre gerar o próximo estado e utilizá-lo como nova entrada.

As repesentações simbólicas apresentadas na Figura 3.4 não são diretamente implementáveis como um sistema digital, como poderia sugerir o desenho. Os aspectos práticos a serem considerados para o necessário processo de "tradução" são explorados no Capítulo 4 e 5. Aqui introduz-se a definição formal destas estruturas.

Definição 3.8 (Grafo de Transição de Estados - STG) *Dada uma máquina de estados finita $\mathcal{M} = \langle I, S, O, \delta, \lambda, S_0 \rangle$, o* ***grafo de transição de estados*** *(STG) que a representa é um grafo dirigido com rótulos $G_{\mathcal{M}} = \langle S, \delta, R^S, R^\delta \rangle$, onde o conjunto de vértices é o conjunto de estados S da FSM, as arestas são definidas pela função de transição da FSM, e R^S e R^δ dependem do tipo de máquina a representar.*

Numa máquina de Moore, $R^S = O$ e $R^\delta = I$ possuem elementos r^S_m e r^δ_n definidos pelas funções próximo estado e de saída, bem como pelas bijeções do grafo rotulado, $f : S \longrightarrow R^S$ e $g : \delta \longrightarrow R^\delta$ da seguinte maneira:

$$\begin{aligned} \delta(i_j, s_k) = r^\delta_n \quad &\Longleftrightarrow \quad g((i_j, s_k), r^\delta_n) = i_j \\ \lambda(s_k) = r^S_m \quad &\Longleftrightarrow \quad f(s_k) = r^S_m. \end{aligned}$$

Numa máquina de Mealy, $R^S = S$ e $R^\delta = I \times O$. R^S não comporta nenhuma informação adicional, a bijeção f sendo a função identidade. R^δ, contudo, possui elementos r^δ_n definidos pelas funções próximo estado e de saída, bem como pela bijeção $g : \delta \longrightarrow R^\delta$ da seguinte maneira:

$$\delta(i_j, s_k) = s_m \wedge \lambda(i_j, s_k) = o_m \quad \Longleftrightarrow \quad g((i_j, s_k), s_m) = (i_j, o_m) = r^\delta_n.$$

Em uma representação gráfica de um STG, normalmente se emprega uma forma distintiva de identificar os vértices associados aos elementos de S_0, tal como indicá-los com uma seta, desenhar vértices com algum padrão diferente dos vértices fora de S_0, etc.

Note-se que a forma do STG para máquinas de Mealy pode ser usada para representar máquinas de Moore, mas o inverso não é possível em geral.

A definição de STG revela a vantagem de modelar funções como relações binárias. Por exemplo, sendo δ acima uma função, uma relação binária, e uma representação alternativa para o grafo da relação, pode-se usar uma notação coerente e compacta (Ver Definição 2.2 para tais convenções).

Definição 3.9 (Tabela de Transição de Estados - STT) *Dada uma máquina de estados finita $\mathcal{M} = \langle I, S, O, \delta, \lambda, S_0 \rangle$, a* ***tabela de transição de estados*** *(STT) que a representa é uma tabela bidimensional que descreve as funções próximo estado e de saída. A forma exata da STT depende do tipo de máquina a representar, havendo formas distintas para representar máquinas de Moore e Mealy.*

Linhas *em uma* STT *correspondem sempre a estados da* FSM*, enquanto existem pelo menos tantas* **colunas** *na* STT *quanto existem entradas distintas, uma associada a cada letra do alfabeto de entrada.*

Em uma máquina de Moore, existe uma coluna adicional, usada para designar a saída associada ao estado correspondente à linha em questão da STT*. Logo, as posições da* STT *contêm informação de próximo estado na intersecção de uma coluna (associada a alguma entrada da* FSM*) com uma linha. A coluna adicional especifica as saídas associadas a cada estado.*

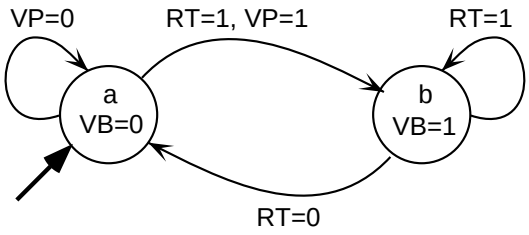
Em uma máquina de Mealy, todas as posições da STT *possuem o mesmo tipo de informação, qual seja, o par de valores associados aos resultados da avaliação das funções próximo estado e de saída, para o par entrada/estado atual associado à intersecção de linha e coluna correspondentes. Assim, uma intersecção associada ao par* (i_j, s_k) *(linha* j *e coluna* k *da* STT*), é preenchido com o resultado de avaliar* $(\delta(i_j, s_k), \lambda(i_j, s_k))$*.*

Tanto STGs quanto STTs permitem representar não-especificações de forma natural, seja de forma implícita, seja de forma explícita. Ver-se-á agora exemplos destas representações aplicados ao problema de síntese lógica do sistema digital de controle da intersecção da BR 2000 com a Picada do Mato.

Exemplo 3.4 (Controle de tráfego síncrono) O STG que representa o comportamento do sistema digital de controle da intersecção já foi ilustrado na Figura 3.3. Em vista do modelo síncrono da Figura 3.4(a), deve-se agora refinar a descrição e definir a estratégia de implementação síncrona do sistema. Em seguida, deve-se proceder à síntese da estrutura lógica desta, ao fim da qual o sistema pode ser construído.

Uma observação inicial que facilita a construção de uma versão síncrona, é que os requisitos de desempenho do problema são muito reduzidos para o estado da arte atual em sistema digitais eletrônicos. Um problema que pode ser identificado no STG da Figura 3.3, é que uma da entradas tem sua informação relevante representada pelo nível lógico (VB), enquanto a outra entrada importa durante transições de valor. Uma implementação puramente síncrona está baseada em elementos de memória controlados todos de forma simultânea pelo mesmo, único sinal de relógio. Uma maneira de uniformizar a interpretação dos sinais de entrada é empregar uma freqüência de relógio muito superior aos 0,01Hz da referência temporal RT. Um relógio de 1MHz neste caso é de baixo custo e largamente suficiente, pois o relógio amostra as entradas 100 milhões de vezes por segundo. Desta forma, uma pequena alteração do STG original permite considerar ambos sinais VP e RT a partir de seus níveis lógicos. A Figura 3.5 mostra o novo STG e o STT correspondente que constituem a especificação síncrona do sistema.

A FSM em questão é então $\mathcal{M} = \langle I = \{00, 01, 10, 11\}, S = \{a, b\}, O = \{0, 1\}, \delta, \lambda, S_0 = \{a\}\rangle$, com δ e λ dados por qualquer das estruturas da Figura 3.5. As variáveis usadas representam as entradas e saídas e serão usadas para



(a) STG

S \ RT/VP	0-	10	11	VB
a	a	a	a	0
b	a	b	b	1

(b) STT

Figura 3.5: STG e STT, a especificação síncrona do sistema digital de controle do cruzamento da BR 2000 com a Picada do Mato.

definir a implementação sob a forma de expressões Booleanas para as funções próximo estado e de saída.

Antes de produzir as equações Booleanas, o problema de codificação de estados deve ser resolvido. Aqui, escolhe-se arbitrariamente a codificação $a = 0$ e $b = 1$ que é correta, embora não comprovadamente ótima. Ao mesmo tempo, associa-se o valor da variável Booleana ST aos estados codificados. O complexo problema determinado pelo passo de codificação será objeto de estudo detalhado no Capítulo 4. A síntese síncrona após este passo é simples, e pode ser obtida por inspeção da STT ou pelo emprego de ferramentas como o mapa de Karnaugh. O resultado pode ser expresso pela equações $\delta((RT, VP), ST) = RT \wedge (VP \vee ST)$ e $\lambda((RT, VP), ST) = ST$. A implementação mediante portas lógicas e elementos biestáveis aparece na Figura 3.6.

Note-se que a implementação da função próximo estado consiste de duas portas lógicas e que a função de saída é implementada por um único fio. Percebe-se também na Figura a estrutura geral de implementação de FSMs como sistemas digitais síncronos: um circuito combinacional com realimentação controlada pelo sinal de relógio e mantida em uma memória.

■

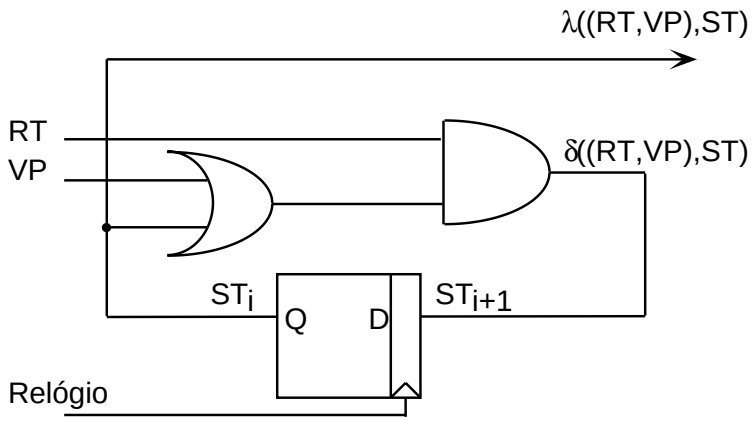


Figura 3.6: Implementação síncrona de um sistema digital no nível lógico de abstração.

3.4 Representações Assíncronas

A síntese automatizada de sistemas digitais síncronos conta com métodos e representações que podem ser facilmente estratificados em níveis de abstração bem distintos. Níveis de abstração mais altos contam com formalismos como linguagens de descrição de hardware e formatos comportamentais abstratos. No nível lógico de abstração, formas de representação síncronas como os STGs e as STTs, definidos acima, possuem interpretações precisas, e são a base para praticamente qualquer método de projeto. O nível elétrico conta com modelos de atraso detalhados usados em pontos avançados do processo de projeto para definir precisamente o desempenho do sistema em termos da máxima freqüência do sinal de relógio aplicável, bem como para garantir que os pressupostos sobre a temporização relativa de sinais são válidos. Esta separação clara de níveis provém, no modelo síncrono, da possibilidade de modelar de forma discreta não apenas os valores das entradas, saídas e sinais internos, mas até mesmo o próprio tempo.

No que diz respeito a implementações assíncronas de sistemas digitais, a situação atual de métodos e mesmo de representações é bem menos definida. Considerações de temporização relativa entre sinais são fundamentais no nível lógico de abstração, pois não é possível discretizar o tempo sem recurso a sinais globais de relógio. Logo, considerações de natureza puramente lógica tais como a modelagem do sistema como um conjunto funcional de portas lógicas são de pouca utilidade *per se*. Além disso, a natureza da atividade seqüencial de sistemas digitais assíncronos requer uma abordagem diferente da noção de *estado*,

pois aquela usada no tratamento de sistemas síncronos é insuficiente. Finalmente, embora sistemas assíncronos sejam vistos como capacitando o emprego de métodos de projeto mais abstratos, ainda não existem formalismos suficientemente estudados a ponto de dizê-los capazes de dar suporte ao processo de projeto destes sistemas em altos níveis de abstração, embora formalismos promissores existam. Alguns destes derivam da área de processamento paralelo, como CSP [100] e OCCAM [178], outros de especificação formal de hardware, tal como Synchronized Transitions [168], e ainda outros da área de linguagens de descrição de hardware propriamente ditas, tal como adaptações de VHDL[175].

A forma de representação geral mais tradicional de sistemas digitais seqüenciais assíncronos é a tabela de fluxo (AFT). Contudo, o modelo de circuito subjacente a esta representação clássica não permite a descrição de alguns dispositivos de grande utilidade, dentre estes uma grande classe de sistemas arbitradores [117].

Uma forma de representação de grande impacto atualmente é o grafo de transição de sinais, ASTG. Apesar de seu uso difundido, algumas questões fundamentais ainda estão por ser respondidas sobre ASTGs. Por exemplo, Lavagno e Sangiovanni-Vincentelli mencionam que não existe nenhum método geral capaz de decidir se um dado ASTG *admite* uma implementação sem problemas de temporização, ou que funcione independente dos atrasos relativos de portas apenas ou de portas e fios em conjunto [117]. Em anos recentes, foi proposta uma definição da *validade* de um ASTG, em função da possibilidade de se implementar um sistema digital assíncrono capaz de reproduzir o comportamento previsto por esta representação. A proposta de Chu em [46] é que o ASTG em questão possua uma rede de Petri [140] subjacente que goze de três propriedades: *segurança*, *vitalidade*, e *livre escolha* (do inglês, *safety*, *liveness*, *free-choice*). Embora estas três condições garantam a implementabilidade do ASTG, elas não são necessárias, e alguns trabalhos [193, 114] mostraram que comportamentos úteis e implementáveis que não podem ser descritos por ASTGs com as três propriedades citadas.

Abaixo realiza-se a revisão necessária do conceito de estado para sistemas assíncronos e define-se de maneira formal AFTs e ASTGs, finalizando com o desenvolvimento de uma implementação assíncrona para o controlador de tráfego do cruzamento da BR 2000 com a Picada do Mato.

3.4.1 Estados em Sistemas Digitais Assíncronos

A noção de estado apresentada no Capítulo 1 e complementada na Seção 3.2 é abrangente, mas informal, uma vez que *história passada do sistema digital* não foi precisamente definida. Contudo, a discretização do tempo em circuitos síncronos provê uma caracterização precisa, tanto de história passada, quanto de quantidade de informação sobre o passado que o sistema pode armazenar. A transição de estado só é possível a intervalos regulares, determinados pelo

relógio, e fora destes períodos, variações de valores de entradas são irrelevantes para gerar transições de estado, embora as saídas possam variar (bem entendido, em máquinas de Mealy apenas). O relógio e a memória interposta entre saídas e entradas do sistema (ver Figura 3.4(a)) impõem assim uma barreira temporal à inflência das entradas sobre o sistema.

Sistemas digitais assíncronos, por outro lado, podem ser afetados a qualquer momento por variações de entradas, e o conceito de estado tem a ver não apenas com os valores armazenados nos laços de realimentação do sistema digital (conforme descrito na Definição 1.2), mas também com:

- o valor instantâneo das entradas;
- a condição de estabilidade dos valores armazenados nos laços de realimentação relativo ao valor atual destes e ao valor atual das entradas.

Com base nesta constatação, pode-se agora definir novos conceitos, relacionados a estados em sistemas digitais seqüenciais assíncronos.

Definições 3.10 (Estados de um sistema assíncrono) *Considere um sistema digital seqüencial assíncrono representado segundo o modelo da Figura 3.4(b). Cada configuração distinta de valores presentes na realimentação (dada por $\delta(i_j, s_k) = s_l$) é denominada um* **estado interno** *do sistema digital assíncrono. O valor instantâneo da entrada i_j é denominado* **estado de entrada** *do sistema digital. A combinação do estado interno e o estado de entrada é o* **estado total** *do sistema digital. O estado interno é dito* **estável** *se, mantido fixo o símbolo de entrada i_j, nenhuma transição de estado puder ocorrer. Se o estado interno for tal que, mesmo fixando a entrada, alguma mudança ocorre, ou pode ocorrer, o estado é dito* **instável**.

Dada esta definição estendida, uma forma de representação síncrona (STG ou STT), todos os assim chamados *estados* nestas são internos e estáveis. Considerações de meta-estabilidade [45] em sistemas digitais síncronos levam a formulações que consideram efeitos transitórios envolvendo estados instáveis, mas afora este fenômeno, os conceitos aqui introduzidos são raramente empregados em projeto de sistemas síncronos.

3.4.2 AFTs e ASTGs

A partir da extensão do conceito de estados visando abranger sistemas assíncronos, pode-se então definir as formas de representação pertinentes.

Definição 3.11 (Tabela de Fluxo - AFT) *Dada uma FSM com a estrutura $\mathcal{M} = \langle I, S, O, \delta, \lambda, S_0 \rangle$, a* **tabela de fluxo** *(AFT) que a representa é uma tabela bidimensional que descreve as funções próximo estado e de saída. A*

	00	01	10	11
1	**1**,00	2,00	**1**,00	4,10
2	**2**,10	**2**,00	**2**,01	3,01
3	1,00	4,01	**3**,01	**3**,01
4	**4**,10	**4**,01	3,01	**4**,10

(a)

	00	01	10	11
1	**1**,00	2,10	**1**,00	4,10
2	**2**,10	3,00	**2**,01	3,11
3	1,01	4,11	**3**,01	**3**,10
4	**4**,10	**4**,01	3,00	**4**,01

(b)

	00	01	10	11
1	**1**,00	2,10	**1**,00	4,10
2	**2**,10	1,00	**2**,01	3,11
3	1,01	4,11	**3**,01	**3**,10
4	**4**,10	**4**,01	3,00	**4**,01

(c)

	00	01	10	11
1	**1**,00	2,10	**1**,00	4,10
2	**2**,10	3,00	**2**,01	3,11
3	1,01	4,11	**3**,01	**3**,10
4	**4**,10	3,01	3,00	**4**,01

(d)

Figura 3.7: Quatro tabelas de fluxo ilustrando os diferentes tipos de comportamentos de FSM possíveis. A estrutura de todas as FSMs da Figura é $\mathcal{M} = \langle\{00,01,10,11\},\{1,2,3,4\},\{00,01,10,11\},\delta,\lambda\rangle$. Posições das AFTs com próximo estado em negrito correspondem a um estado total estável. (a) é uma AFT do tipo SOC, onde toda coluna possui pelo menos um estado total estável associado, e a partir de qualquer estado, recebendo qualquer entrada, se chega a um estado total estável com apenas uma transição de valores de saída. (b) é uma AFT do tipo MOC, devido à estrutura da segunda coluna, mas não apenas. Estando no estado total $(2,10)$ e havendo uma transição de entrada de 10 para 11, passa-se diretamente para o estado interno 3, mas produz-se duas transições de saída, de 01 para 11 (produzida no estado total instável $(2,11)$), e daí para 10, no estado estável. Note-se que esta última situação jamais pode ocorrer na AFT (a). (c) e (d) são AFTs do tipo UOC. Em (c), toda entrada pode fazer o sistema atingir um estado total estável, embora isto não seja garantido em todas as situações para a entrada 01. Em (d), o sistema certamente entra em oscilação quando recebe a entrada 01, pois nenhum estado total estvel pode ser obtido com esta entrada.

forma exata da STT *depende do tipo de máquina a representar, havendo formas distintas para representar máquinas de Moore e Mealy.*

Linhas *em uma* AFT *correspondem sempre a estados da* FSM*, enquanto existem pelo menos tantas* **colunas** *na* AFT *quanto existem entradas distintas, uma associada a cada letra do alfabeto de entrada.*

Em uma máquina de Moore, existe uma coluna adicional, usada para designar a saída associada ao estado correspondente a linha em questão da AFT*. Logo, as posições da* AFT *contêm informação de próximo estado na intersecção de uma coluna (associada a alguma entrada da* FSM*) com uma linha. A coluna*

adicional especifica as saídas associadas a cada estado.

Em uma máquina de Mealy, todas as posições da AFT *possuem o mesmo tipo de informação, qual seja, o par de valores associados aos resultados da avaliação das funções próximo estado e de saída, para o par entrada/estado atual associado à intersecção de linha e coluna correspondentes. Assim, uma intersecção associada ao par* (i_j, s_k) *(linha* j *e coluna* k *da* STT*), é preenchido com o resultado de avaliar* $(\delta(i_j, s_k), \lambda(i_j, s_k))$.

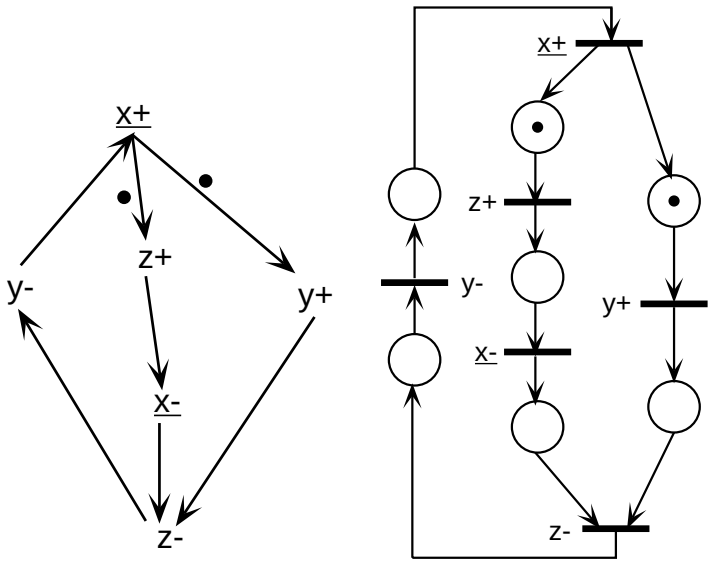
Um ***estado total estável*** *da* AFT *corresponde a uma posição da tabela onde a linha está associada a um estado* s_k*, a coluna está associada a um símbolo de entrada* i_j *e tem-se* $\delta(i_j, s_k) = s_k$*. Todas as demais situações correspondem a estados totais instáveis.*

Pode-se classificar AFT*s, e portanto sistemas digitais assíncronos, com relação ao possível número de transições de valores de saída gerados por exatamente uma transição de valor na entrada. Se no máximo uma transição de saída é possível nesta situação, a* AFT *é dita de* ***mudança de saída única*** *(em inglês,* single-output-change *ou* SOC*). Se mais de uma transição de saída é possível, mas o número de transições é sempre limitado, a* AFT *é dita de* ***mudanças de saída múltiplas*** *(em inglês,* multiple-output-change *ou* MOC*). Finalmente, se mais de uma transição de saída é possível, mas não é possível determinar o número máximo destas, a* AFT *é dita de* ***mudanças de saída ilimitadas*** *(em inglês,* unbounded-output-change *ou* UOC*).*

A definição de AFT indica que esta possui exatamente a mesma estrutura de STTs, exceto pela informação de estabilidade de estados totais. A classificação de AFTs e de sistemas digitais assíncronos em SOC, MOC, e UOC é fundamental para analisar a possibilidade de realizar um sistema digital assíncrono, sendo a primeira classe, SOCs, a de maior facilidade de tratamento, e a mais limitada. A Figura 3.7 ilustra os diferentes tipos de AFT.

ASTGs são um tipo de redes de Petri. Estas são um modelo usado amplamente para descrever sistemas concorrentes, pois possuem uma semântica simples e intuitiva, que é capaz de capturar diretamente conceitos importantes tais como *causalidade*, *concorrência* e *conflito* entre eventos. Freqüentemente, representa-se redes de Petri como um tipo especial de grafo dirigido, onde existem dois tipos distintos de vértices e as arestas sempre conectam vértices de tipos diferentes, no que se define como um ***grafo bipartido*** (do inglês, *bipartite graph*).

Definição 3.12 (Redes de Petri) *Uma* ***rede de Petri*** *(*PN*) é uma tripla* $\mathcal{P} = \langle T, L, F \rangle$*, onde* T *é o conjunto não-vazio de* ***transições****,* L *é o conjunto não-vazio de* ***lugares*** *e* $F \subseteq (T \times L) \cup (L \times T)$ *é a* ***relação de fluxo*** *entre transições e lugares. Uma* ***marcação*** *é uma função* $m : L \longrightarrow \{0, 1, 2, \ldots\}$*, onde* $m(l)$ *é o número de* ***marcas*** *em* l *sob a marcação* m*. Uma* PN ***marcada*** *é uma quádrupla* $\mathcal{P} = \langle T, L, F, m_0 \rangle$*, onde* m_0 *denota a* ***marcação inicial****.*



(a) ASTG (b) Rede de Petri

Figura 3.8: Um exemplo de ASTG e a rede de Petri subjacente. O sinal sublinhado corresponde a uma entrada primária. (a) Lugares são representados implicitamente quando possuem apenas uma aresta incidente, transições são designadas pelo sinal que transiciona e um indicador de tipo de borda, + sendo uma borda de subida do sinal, − uma borda de descida. (b) Lugares são representados como círculos vazados, transições como barras horizontais. A marcação inicial é designada por círculos sólidos em ambas representações. Existe na marcação inicial duas transições aptas a disparar, $z+$ e $y+$, as que possuem todos os respectivos lugares de entrada marcados.

Definição 3.13 (Grafo de Transição de Sinais - ASTG) *Um (ASTG), ou* ***grafo de transição de sinais*** *é uma quádrupla* $\mathcal{G} = \langle \mathcal{P}, I, O, \Delta \rangle$*, onde* $\mathcal{P}$ *é uma rede de Petri marcada,* I *e* O *são conjuntos disjuntos finitos de* ***entradas*** *e* ***saídas****, respectivamente, e* $\Delta : T \longrightarrow (I \cup O) \times \{+, -\}$ *rotula cada transição de* $\mathcal{P}$ *com uma transição de sinal. Um ASTG é* ***autônomo*** *se ele não possui nenhum sinal de entrada, ou seja, se* $I = \emptyset$*.*

A idéia de marcação em ASTGs está associada à evolução de transições de estado do sistema digital que ela descreve. A marcação inicial corresponde a um estado inicial do sistema. A regra básica que rege o ***disparo de uma transição***, ou seja a passagem de uma marcação a outra, é que uma transição

Tabela 3.2: AFT parcial, que descreve a situação em que não há tráfico na Picada do Mato.

$I \rightarrow$	x_1x_2			
$S \downarrow$	00	01	11	10
1	**1**, 0			2, 0
2	1, 0			**2**, 0

consome marcas nos seus lugares de entrada e produz marcas nos seus lugares de saída, mas apenas se todos os seus lugares de entrada estiverem simultaneamente marcados. A Figura 3.8 mostra um exemplo de ASTG com a rede de Petri subjacente a esta.

3.4.3 Uma Implementação Assíncrona

Ilustrando as representações mostradas na Seção anterior, procede-se agora à implementação assíncrona do sistema de controle do cruzamento da BR 2000 com a Picada do Mato.

Exemplo 3.5 (Controle de tráfego assíncrono) O processo de análise para a implementação assíncrona deve levar em conta a redefinição do conceito de estado provido pela Definição 3.10. Inicialmente, procura-se obter uma AFT representando o comportamento esperado do sistema.

Para facilitar a análise, procura-se obter uma AFT com um estado estável por linha apenas. Tais AFTs são denominadas AFT***s primitivas***. Mais tarde, procedimentos de minimização de estados podem ser aplicados para reduzir a quantidade adicional de linhas gerada no processo. Assume-se aqui as mesmas variáveis de entrada (RT e VP), de saída (VB) e a mesma codificação do caso síncrono.

Suponha que a referência temporal possui um valor $RT = 0$, nenhum veículo está presente na Picada, ou seja $VP = 0$, e que esta situação perdura há alguns minutos. Logo, o sinal deve estar verde para a BR e vermelho para a Picada, ou seja $VB = 0$. Chamando-se este estado interno de 1, o rótulo do estado total estável mencionado é $(1, 00)$, onde o primeiro dígito indica o estado interno e os dois últimos o estado de entrada, na ordem já usada antes, $RT\ VP$. Uma vez que este estado é estável, o código de próximo estado deve ser indicado em negrito na AFT (ver a Tabela 3.2).

Enquanto o sistema está neste estado total, suponha que RT muda para 1. O estado total muda imediatamente para $(1, 10)$, VB permanece em 0 e o próximo estado deve ser algo diferente de 1 (pois assumiu-se que apenas um estado total estável por linha da AFT). Chame-se este novo estado de 2. Logo, a posição $(1, 10)$ da AFT deve conter $(2, 0)$. Considere-se agora o estado total

Tabela 3.3: AFT final para implementar o sistema digital assíncrono de controle do cruzamento da BR 2000 com a Picada do Mato.

$I \rightarrow$	x_1x_2			
$S \downarrow$	00	01	11	10
1	**1**, 0	3, 0	7, 0	2, 0
2	1, 0	3, 0	7, 0	**2**, 0
3	4, 0	**3**, 0	6, 1	5, 1
4	**4**, 0	3, 0	6, 1	5, 1
5	1, 0	3, 0	6, 1	**5**, 1
6	1, 0	3, 0	**6**, 1	5, 1
7	4, 0	3, 0	**7**, 0	8, 0
8	4, 0	3, 0	7, 0	**8**, 0

$(2, 10)$. O estado interno corresponde a uma situação na qual nenhum $VP = 1$ ocorreu por longo tempo, logo VB deve permanecer em 0 pelo menos por toda a duração do intervalo em que $RT = 1$. Logo, este também é um estado total estável, com a AFT apresentando $(2, 0)$ nesta posição.

Suponha agora que RT volte a 0 enquanto o sistema se encontra no estado total $(2, 10)$. A situação volta a ser a mesma do início, e o conteúdo da tabela na posição $(2, 00)$ deve ser $(1, 0)$. Este fragmento da AFT é mostrado na Tabela 3.2, e indica precisamente o que ocorre em regime estacionário, à medida que a referência temporal evolui e nenhum veículo se apresenta na Picada do Mato. Os estados totais do sistema permanecem circulando ao longo de $(1, 00), (1, 10), (2, 10), (2, 00), (1, 00), \ldots$, com VB fixo em 0.

A construção da AFT continua preenchendo os vazios nas linhas 1 e 2, adicionando novas linhas e novos estados que sejam necessários, até que o processo (espera-se) pare com a tabela completa.

O passo seguinte seria inserir a informação $(3, 0)$ na posição $(1, 01)$ e $(3, 0)$ na posição $(3, 01)$ (indicando chegada de veículo na Picada, mas ainda não ativando uma mudança de sinal, pois $RT \neq 1$). Note-se que após cada transição de entrada, o sistema move-se de seu estado estável atual para um estado instável na mesma linha, e daí para um estado total estável na coluna correspondente à nova entrada. Este comportamento é característico de AFTs primitivas.

Após uma análise completa da situação, obtém-se a AFT primitiva completa mostrada na Tabela 3.3.

Como um exemplo adicional de interpretação da AFT completa, considere que existe uma feira agrícola acontecendo, e que portanto um grande tráfico está ocupando a Picada do Mato. Neste caso, apenas as duas colunas do meio da Tabela 3.3 estariam sendo usadas, supondo que o sinal do sensor de veículo na Picada estivesse sempre ativado. Os estados totais visitados seriam, ciclicamente, $(\mathbf{3}, 01), (3, 11), (\mathbf{6}, 11), (6, 01)$, o sistema permanecendo a quase

totalidade dos 70 segundos do ciclo de RT no primeiro estado total estável desta série, $(\mathbf{3}, 01)$, e a quase totalidade dos 30 segundos restantes no segundo estado total estável, $(\mathbf{6}, 11)$.

Uma inspeção da Tabela 3.3 permite identificar, ainda que intuitivamente, que existem redundâncias nesta. A Tabela 3.4 mostra uma AFT equivalente a tabela completa, com apenas metade do número de linhas. O leitor pode verificar a equivalência deste comportamento em relação à Tabela 3.3. O procedimento sistemático para minimização de estados que gerou a Tabela 3.4 é discutido no Capítulo 4 para sistemas síncronos, sendo aplicável igualmente a AFTs. Note que os estados da nova AFT não guardam necessariamente relação com os de mesma denominação na AFT original.

Antes de realizar a síntese do sistema digital assíncrono que corresponde à Tabela 3.4 é necessário codificar os estados internos da AFT. Na Tabela 3.4 existe uma sugestão de codificação usando o número mínimo absoluto de bits, 2. Note-se que existem $4! = 24$ opções distintas de codificação nestas condições, e que esta não é necessariamente a melhor, nem sequer uma que se saiba garantir a ausência de fenômenos indesejáveis associados a sistemas assíncronos tais como corridas e transitórios. Esta foi uma escolha aleatória, visando ir até o fim do processo de síntese. Mais detalhes sobre os requisitos de corretude e otimização a serem atendidos pelo projeto assíncrono aparecerão no Capítulo 5.

Dada a codificação de estados, pode-se aplicar técnicas de minimização como o mapa para obter as equações que definem a lógica seqüencial, ou seja as equações de próximo estado e de saída. Como antes, VB é a variável dependente que representa a função de saída. Para o próximo estado, duas variáveis são necessárias. Visando salientar a identidade das entradas de estado atual e as respectivas saídas de próximo estado, usa-se as variáveis ST_1 e ST_2 para representar as últimas, em função das entradas primárias RT e VP e do estado atual, dado pelas variáveis st_1 e st_2. Aplicando mapas de Karnaugh, por exemplo, pode-se obter as seguintes equações:

$$\begin{aligned} VB &= (RT \wedge \overline{st_1} \wedge st_2) \vee (RT \wedge st_1 \wedge \overline{st_2}), \\ ST_1 &= (RT \wedge VP \wedge \overline{st_1} \wedge \overline{st_2}) \vee (RT \wedge \overline{st_1} \wedge st_2) \vee (RT \wedge st_1), \\ ST_2 &= (\overline{RT} \wedge VP) \vee (st_1 \wedge st_2) \vee (\overline{RT} \wedge st_2) \vee (RT \wedge VP \wedge \overline{st_1} \wedge \overline{st_2}). \end{aligned}$$

Métodos de projeto tradicionais de sistemas assíncronos pressupõem que st_1 e st_2 nada mais são que versões defasadas no tempo dos sinais ST_1 e ST_2, respectivamente. Para obter este efeito, pode-se adicionar por exemplo, pares de inversores na parte externa do laço de realimentação, caso o atraso dos fios não seja suficiente por si para garantir o atraso necessário ao bom funcionamento da implementação. Compare-se as equações acima, que contam com um total de 25 literais à parte combinacional da implementação síncrona, com 4 literais. Mesmo considerando a complexidade da memória que armazena o estado na

Tabela 3.4: AFT final reduzida para implementar o sistema digital assíncrono de controle do cruzamento da BR 2000 com a Picada do Mato. Também se mostra uma proposta de codificação de estados usando as variáveis st_1 e st_2.

$I \rightarrow$	x_1x_2					
$S \downarrow$	00	01	11	10	st_1	st_2
1	**1**, 0	2, 0	4, 0	**1**, 0	0	0
2	**2**, 0	**2**, 0	3, 1	3, 1	0	1
3	1, 0	2, 0	**3**, 1	**3**, 1	1	0
4	2, 0	2, 0	**4**, 0	**4**, 0	1	1

versão síncrona, a diferença é grande. Esto fato explica pacialmente o até então pequeno sucesso de implementações assíncronas de sistemas digitais.

■

3.5 Conclusões do Capítulo

Representações seqüenciais têm sua origem em modelos abstratos derivados das teorias da computação e de linguagens formais. As representações seqüenciais possuem formas gráficas e tabulares que congregam características destes modelos abstratos de um lado e de abstrações físicas de sistemas digitais de outro. A conjunção de modelos e abstrações físicas possibilitam o emprego destas representações como base de métodos de projeto de sistemas digitais.

Representações síncronas e assíncronas diferem em muitos pontos, tais como complexidade, nível de abstração, e na maneira de considerar o conceito de estado.

Apresentaram-se aqui apenas algumas das mais consagradas formas de representação no nível lógico. Para as formas seqüenciais sobretudo, existe uma gama ampla de representações alternativas. No entanto, as formas apresentadas permitem descrever a grande maioria dos comportamentos viáveis para sistemas seqüenciais no nível lógico de abstração.

3.6 Perspectiva Histórica e Bibliografia

O estudo de caso adotado ao longo do Capítulo foi adaptado a partir do exemplo disponível no livro de Unger [182] sobre síntese de circuitos assíncronos. Os conceitos formais vinculados à máquinas de estados finitas apoiaram-se nas exposições de Kohavi em [115] e Zahnd em [197], este último um livro de beleza formal ímpar, aliada a uma clareza de conceitos incomum. Boa parte da discussão sobre AFTs originou-se também do já citado trabalho de Unger, enquanto que a definição de ASTGs usada aqui, bem como todo um corpo de

métodos de projeto assíncrono baseado neles pode ser encontrado na obra de Lavagno e Sangiovanni-Vincentelli [117].

3.7 Resumo do Capítulo

A partir de um estudo de caso prático, desenvolveu-se aqui um conjunto de representações clássicas para sistemas seqüenciais, constrastanto as formas de representação síncronas com as formas assíncronas. Alguns resultados importantes, como a equivalência entre autômatos finitos e linguagens regulares foram mencionados, embora não desenvolvidos. Viu-se ainda como as formas assíncronas de representação são em princípio mais gerais, podendo representar igualmente sistemas assíncronos. Na maioria das situações, porém, não há motivo para utilizar estas formas mais complexas ao lidar com sistemas assíncronos.