SÍNTESE DE SISTEMAS DIGITAIS UTILIZANDO
TÉCNICAS EVOLUTIVAS

Sérgio Granato de Araújo

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA ELÉTRICA.

Aprovada por:

_____
Prof. Aloysio de Castro Pinto Pedroza, Dr.

_____
Prof. Antônio Carneiro de Mesquita Filho, Dr.d'État

_____
Profa. Luci Pirmez, D.Sc.

_____
Prof. Valmir Carneiro Barbosa, Ph.D.

_____
Prof. Marco Aurélio Cavalcanti Pacheco, Ph.D.

_____
Prof. Leandro dos Santos Coelho, D.Sc.

RIO DE JANEIRO, RJ – BRASIL
JULHO DE 2004

ARAÚJO, SÉRGIO GRANATO DE

Síntese de Sistemas Digitais Utilizando Técnicas Evolutivas [Rio de Janeiro] 2004

XV, 138 p. 29,7 cm (COPPE/UFRJ, D.Sc., Engenharia Elétrica, 2004)

Tese - Universidade Federal do Rio de Janeiro, COPPE

1. Sistemas Digitais

2. Algoritmos Evolutivos

3. Síntese de Alto Nível

4. Protocolos de Comunicação

I. COPPE/UFRJ II. Título (série)

À Indiara, Igor e Enzo.

Agradecimentos

Aos meus pais Djalma (*in memoriam*) e Isa, pelo amor, apoio e constante incentivo. À minha esposa Indiara, pelo amor, apoio e compreensão pelo tempo subtraído do convívio familiar. Aos meus filhos Igor e Enzo, pelo eterno carinho.

Aos profs. Aloysio de Castro P. Pedroza e Antônio Carneiro de Mesquita F., meus orientadores, pelo incentivo, orientação, reconhecimento e amizade durante este trabalho e desde o início de minha graduação. Aos profs. Valmir, Luci Pirmez, Marco Aurélio e Leandro, pela presença na banca e contribuição à tese. Aos profs. Valmir, Luci Pirmez e Marco Aurélio, pelas sugestões propostas no exame de qualificação.

O autor é extremamente grato ao eng. José M. Aravena Z., pela amizade, apoio operacional e confiança depositada. A concretização deste trabalho, assim como o do curso de mestrado, deveu-se muito ao seu constante incentivo.

Aos profs. do GTA/UFRJ pelas instruções e amizade. Ao Márcio, Marcial e a todos colegas de doutorado e mestrado das turmas de 2000, 2001, 2002 e 2003, pelo convívio e amizade. Aos colegas extraclasse Vahé e P. Filadelfo, pelo apoio e amizade.

Aos meus irmãos José Vicente, Marcelo e Paulo pelo apoio, especialmente na revisão de artigos e do texto da tese. Aos demais irmãos André, Beto, Maria Inês e Adriana, e respectivas famílias, pelo apoio e convívio nessa jornada. À família de minha esposa, especialmente à dona Diva, pelo apoio à minha família o período de meu afastamento.

À Fundação Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) pelo financiamento da pesquisa de tese. À Escola de Engenharia Elétrica da UFG pela licença concedida para a realização destes estudos. Ao pessoal da secretaria do PEE/COPPE e da EEE/UFG, pelo suporte administrativo. À TIC projetos, representada pelo eng. Luiz Sérgio, pelo apoio e suporte operacional.

A todos que colaboraram de alguma forma para que este trabalho fosse concluído e, por falha de minha memória, não foram citados.

O meu muito obrigado a todos vocês.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências (D.Sc.)

SÍNTESE DE SISTEMAS DIGITAIS UTILIZANDO TÉCNICAS EVOLUTIVAS

Sérgio Granato de Araújo

Julho/2004

Orientadores: Aloysio de Castro Pinto Pedroza
Antônio Carneiro de Mesquita Filho

Programa: Engenharia Elétrica

Esta tese investiga a aplicação de técnicas evolutivas no desenvolvimento de sistemas digitais combinacionais e seqüenciais. A computação evolutiva compreende um conjunto de técnicas de busca eficazes na resolução de problemas cujos espaços de busca têm caráter combinatorial. No domínio dos circuitos digitais combinacionais é investigada a implementação automática da funcionalidade do sistema, assim como a qualidade da mesma em termos de área. Para tal, utilizou-se programação genética orientada à gramática de um subconjunto da linguagem VHDL, que contou com a definição de uma função objetivo com múltiplos parâmetros. No domínio dos sistemas seqüenciais, a modelagem de tais sistemas a partir de eventos observáveis é investigada. Em uma primeira etapa, são propostas metodologias para a modelagem de máquinas seqüenciais síncronas a partir de seqüências parciais de entrada e saída. Em seguida, essas metodologias são estendidas para serem aplicadas ao projeto de protocolos de comunicação, por ser essa atividade ainda muito dependente da habilidade do projetista. Nesse caso, o protocolo é assumido de natureza seqüencial. Duas metodologias são propostas: a síntese de protocolos, quando é fornecida a especificação do serviço do protocolo na entrada; e o projeto de protocolos a partir de cenários de interação. As metodologias de projeto de protocolos de comunicação apresentadas neste trabalho propõem simplificar ou mesmo eliminar etapas do processo convencional de desenvolvimento desses sistemas, reduzindo o grau de detalhamento do projeto e, portanto, elevando o nível de abstração que o projetista tem a considerar.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

DIGITAL SYSTEM SYNTHESIS USING EVOLUTIONARY TECHNIQUES

Sérgio Granato de Araújo

July/2004

Advisors: Aloysio de Castro Pinto Pedroza
Antônio Carneiro de Mesquita Filho

Department: Electrical Engineering

This thesis investigates the use of evolutionary techniques in the development of combinational and sequential digital systems. Evolutionary computation concerns to a set of efficient search techniques to problem solutions in which search spaces are combinatorial. In the combinational digital circuits domain, the automatic implementation of the system functionality is investigated, as well as the implementation quality in terms of area. To this purpose, grammar-guided genetic programming of a subset of the VHDL language was used, leading to the development of a multi-parameter objective function. In the sequential systems domain, the modeling of such systems from observable events is investigated. In a first step, methodologies for the modeling of synchronous sequential machines from partial input/output sequences are proposed. Next, these methodologies are extended to be applied in the communication protocols design, as this task demands a lot from the designer abilities. In this case the protocol is assumed sequential. Two methodologies are proposed: the protocol synthesis, when protocol service specifications are supplied at input; and the protocols design coming from interaction scenarios. The communication protocols design methodologies presented in this work intend to simplify or even eliminate steps from the conventional synthesis process of these systems, reducing the design detail degree and, therefore, raising the abstraction level the designer needs to consider.

Lista de Acrônimos

	Pág.	
ADF:	23	*Automatically Defined Function*;
AG:	11	Algoritmo Genético;
AIMGP:	16	*Automatic Induction of Machine code with GP*;
AM:	8	Aprendizado de Máquina;
ASIC:	35	*Application Specific Integrated Circuit*;
BNF:	22	*Backus-Naur Form*;
CAD:	30	*Computer-Aided Design*;
CCS:	79	*Calculus of Communicating Systems*;
CFG:	24	*Context-Free Grammar*;
CFG-GP:	24	*Context-Free Grammar Genetic Programming*;
CI:	34	Circuito Integrado;
CWB:	80	*Concurrency Workbench*;
DES:	75	*Discrete-Event Systems*;
DNA:	16	*Deoxyribonucleic Acid*;
EDA:	35	*Electronic Design Automation*;
EHW:	31	*Evolvable Hardware*;
EP_E:	85	Entidade de Protocolo (Lado Emissor);
EP_R:	85	Entidade de Protocolo (Lado Receptor);
$EPRO_E$:	85	Especificação do Protocolo (Lado Emissor);
$EPRO_R$:	85	Especificação do Protocolo (Lado Receptor);
ES:	79	Especificação do Serviço do Protocolo;
ES_E:	79	Especificação do Serviço do Protocolo (Lado Emissor);
ES_G:	79	Especificação Global do Serviço do Protocolo;
ES_R:	79	Especificação do Serviço do Protocolo (Lado Receptor);
FPGA:	32	*Field Programmable Gate Array*;
FSM:	52	*Finite-State Machine*;
FSMD:	34	*Finite-State Machine Datapath*;
G^3P:	24	*Grammar-Guided Genetic Programming*;

LISTA DE ACRÔNIMOS

	Pág.	
GADS:	27	*Genetic Algorithm for Deriving Software*;
GE:	27	*Grammatical Evolution*;
GP:	12	*Genetic Programming*;
GPK:	25	*Genetic Programming Kernel*;
GPM:	25	*Genotype-Phenotype Mapping*;
HDL:	35	*Hardware Description Language*;
HLS:	34	*High-Level Synthesis*;
hMSC:	98	*high-level MSC*;
IA:	7	Inteligência Artificial;
ILP:	23	*Inductive Logic Programming*;
LALR(1):	26	*Look Ahead Left Recursive - look ahead one symbol*;
LOGENPRO:	23	*LOgic grammar-based GENetic PROgramming*;
LOTOS:	78	*Language Of Temporal Ordering Specification*;
MEF:	34	Máquina de Estados Finitos;
MSC:	97	*Message Sequence Chart*;
PADO:	17	*Parallel Algorithm Discovery and Orchestration*;
PDP:	9	*Parallel Distributed Processing*;
PDU:	85	*Protocol Data Unit*;
PG:	11	Programação Genética;
PNR:	79	*Petri Net model with Registers*;
RTL:	34	*Register-Transfer Level*;
SAP:	77	*Service Access Point*;
SCCS:	130	*Synchronous CCS*;
SCED:	100	*SCenario EDitor*;
ST:	130	Seqüência de Treinamento;
STG:	87	*State-Transition Graph*;
TCCS:	68	*Temporal CCS*;
TDF:	130	Técnica de Descrição Formal;
TEFSM:	81	*Time Extended Finite-State Machine*;
UIO:	79	*Unique Input/Output Sequences*;

LISTA DE ACRÔNIMOS

	Pág.	
VHDL:	89	*Very High-Speed Integrated Circuit (VHSIC) HDL*.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Conceituação Inicial

A *computação evolutiva* consiste no projeto e na análise de algoritmos probabilísticos inspirados em princípios da seleção natural e suas variações [1]. A seleção natural, teoria introduzida pelo naturalista C. Darwin [2] como uma visão unificada para a origem e evolução de organismos na natureza, é a força propulsora que distingue os sistemas biológicos dos sistemas físicos e químicos. Ela pode ser definida como o mecanismo que relaciona *cromossomos* com a eficiência das entidades que eles representam, permitindo que *organismos* mais eficientes se *reproduzam* mais freqüentemente que outros menos eficientes. A motivação para o uso de técnicas evolutivas apóia-se na constatação de que se a natureza tem produzido formas de vida altamente complexas, como seres humanos, seu princípio pode ser utilizado para construir máquinas "inteligentes".

Todo algoritmo evolutivo é composto por um conjunto de elementos fundamentais, a despeito das ramificações que a computação evolutiva possa ter. Os mais importantes dentre estes elementos são:

- Um grupo de indivíduos traduzido por uma *população* de soluções potenciais;
- Uma função de avaliação a ser otimizada (maximizada ou minimizada) que atribui uma *aptidão* (*fitness*) a cada solução potencial;
- Um mecanismo de seleção baseado na aptidão das soluções potenciais;
- Uma fonte de variação composta por operadores inspirados na natureza que permitam alterar as soluções potenciais.

Os três últimos elementos acima citados, conjuntamente, realizam reprodução, im-

põem variáveis aleatórias, promovem competição e realizam seleção em uma dada população. Todas essas operações são executadas de forma iterativa em computadores para simular a evolução de uma população de soluções potenciais através de gerações.

As técnicas evolutivas vêm assumindo lugar de destaque entre os sistemas de aprendizado automático, beneficiadas pelo alto desempenho dos sistemas computacionais. Uma de suas vantagens significativas está na possibilidade de resolver problemas pela simples descrição matemática por meio de uma função objetivo, a qual exprime o que se quer obter na solução, não havendo necessidade de indicar os procedimentos até o resultado. Em especial, as técnicas evolutivas aplicam-se a sistemas de variáveis discretas, onde soluções determinísticas não estão disponíveis. A principal desvantagem do uso de técnicas evolutivas é seu custo computacional elevado decorrente da manipulação e, principalmente, da avaliação de populações de soluções candidatas.

Os algoritmos evolutivos distinguem-se principalmente pela forma com que representam as soluções [3]. O algoritmo genético e a programação genética são as principais instâncias da computação evolutiva. Enquanto o primeiro, na sua forma simples, representa soluções através de cadeias de bits de tamanho fixo, a programação genética utiliza genótipos de tamanhos variados possibilitando codificações mais complexas. O algoritmo genético encontra aplicação em sistemas de otimização onde o problema é complexo e/ou não linear. A programação genética, por permitir que o tamanho dos cromossomos varie, é mais adequada para criar estruturas.

A programação genética enfatiza características desejáveis, tais como a utilização de representações de procedimentos (programas de computador), a capacidade de descobrir e explorar as características intrínsecas do domínio de aplicação e a flexibilidade de adaptar a forma e complexidade de modelos instruídos [4]. Desde sua concepção [5], a programação genética vem sendo utilizada com sucesso para gerar conhecimento, obtendo, por vezes, resultados superiores aos alcançados por meio de técnicas convencionais [6].

Este trabalho investiga a aplicação de algoritmos evolutivos como ferramenta de projeto de sistemas digitais. Sistemas digitais são sistemas que processam *sinais digitais*, isto é, sinais que assumem um número finito de valores discretos. O projeto de sistemas digitais compreende duas grandes áreas do conhecimento: sistemas combinacio-

nais e sistemas seqüenciais. No primeiro caso, as saídas do sistema dependem unicamente dos dados de entrada em um determinado instante, enquanto que no segundo caso o comportamento do sistema depende de uma seqüência de entradas durante um período de tempo.

Os sistemas digitais, sejam eles implementados em *hardware*, *software*, ou como uma combinação de ambos, são a base de grande parte da tecnologia hoje disponível, quais sejam: computadores, sistemas de automação, sistemas de informação, redes de telecomunicações e *internet*. Devido à complexidade que esses sistemas assumem, novas técnicas de projeto têm sido pesquisadas com objetivo de elevar o grau de automatização e prover, assim, um ambiente de projeto menos dependente da habilidade do projetista.

1.2 Objetivos

Este trabalho tem como objetivo principal investigar o emprego de algoritmos evolutivos no desenvolvimento de sistemas digitais combinacionais e seqüenciais, apresentando metodologias que utilizam este paradigma de aprendizado para tratar classes de problemas dentro desses dois domínios.

Como objetivo específico, este trabalho examina o emprego de técnicas evolutivas no projeto e desenvolvimento de sistemas de eventos discretos, como os protocolos de comunicação, por ser essa atividade ainda muito dependente da habilidade do projetista.

1.3 Contribuições

A *primeira contribuição* deste trabalho é a proposta de uma metodologia de síntese de circuitos digitais combinacionais através de programação genética e de ferramentas de síntese de alto nível [7, 8, 9]. A metodologia garante a qualidade do circuito sintetizado através de uma otimização no uso de recursos a partir de uma descrição feita em

alto nível de abstração.

A *segunda contribuição* deste trabalho é a proposta de metodologias que utilizam técnicas evolutivas para sintetizar sistemas seqüenciais a partir de seqüências de entrada e saída observáveis. Particularmente, são desenvolvidas técnicas para gerar máquinas de estados finitos com número mínimo de estados e para tratar o problema de convergência prematura [10]. A *terceira contribuição* do presente trabalho consiste na extensão das metodologias acima propostas para auxiliar o projeto de protocolos de comunicação tanto em abordagens sintéticas [11, 12] quanto em abordagens analíticas [13], com a inclusão de heurísticas para gerar seqüências de treinamento a partir do modelo de serviço do protocolo e de cenários de interação.

1.4 Estrutura do Trabalho

Este trabalho encontra-se estruturado em seis capítulos. O capítulo 2 descreve os principais paradigmas de aprendizado de máquina, um importante ramo da inteligência artificial, discutindo as principais abordagens empregadas na aquisição automática de conhecimento. São confrontados métodos tradicionais de inteligência artificial e novos paradigmas de aprendizado de máquina, tais como as redes neurais e os modelos emergentes e sociais. Conceitos e parâmetros utilizados na computação evolutiva são apresentados, assim como uma descrição detalhada de programação genética. Por fim, faz-se uma revisão literária de abordagens de programação genética orientada à gramática.

O capítulo 3 apresenta uma metodologia a ser aplicada à síntese de circuitos digitais combinacionais, isto é, circuitos lógicos que não utilizam redes de realimentação. A metodologia provê um ambiente para o projeto automático de circuitos lógicos considerando a otimização de recursos, tal como a redução do número de células lógicas utilizadas para implementar a funcionalidade desejada. Esta metodologia tem como principal característica o uso de uma linguagem de descrição de *hardware* para descrever o comportamento das soluções candidatas. Para testar o desempenho da metodologia, foi sintetizada em *hardware* uma função que produz uma aproximação para a Distância Euclidiana entre dois pontos em um espaço bidimensional.

O capítulo 4 apresenta metodologias a serem aplicadas à síntese de sistemas seqüenciais cuja especificação é apresentada na forma de seqüências parciais de entrada e saída. As metodologias geram na saída uma máquina de estados finitos que modela o sistema quando se tem æsso apenas aos eventos de interface do mesmo. Essas metodologias foram aplicadas à síntese de circuitos de lógica seqüencial síncrona de pequeno porte, tais como contadores e detectores de seqüência.

O capítulo 5 estende as metodologias descritas no capítulo 4 para tratar o projeto de protocolos de comunicação. Duas abordagens são apresentadas. A primeira refere-se à síntese de protocolos de comunicação, cujo objetivo é gerar uma especificação do protocolo a partir de uma outra especificação em nível mais elevado, isto é, a especificação de serviço do protocolo. A segunda abordagem concentra-se no projeto de protocolos de comunicação a partir de descrições semiformais, como os cenários de interação. São apresentados e discutidos dois experimentos que geram protocolos a partir de diferentes especificações iniciais.

Por fim, o capítulo 6 apresenta as conclusões do trabalho de pesquisa desenvolvido. São apresentadas também as possíveis derivações da presente linha de pesquisa.

Capítulo 2

Fundamentos

2.1 Introdução

Os sistemas de engenharia que estão sendo atualmente projetados tendem a ser cada vez mais complexos, em especial os sistemas eletrônicos, de computação e de teleprocessamento. Estes sistemas são difíceis de serem desenvolvidos através de metodologias que partem de descrições estruturais ou físicas do projeto. A elevação do grau de abstração no projeto de tais sistemas torna-se essencial, tendo sido alvo de inúmeros trabalhos de pesquisa [14, 15].

O desenvolvimento de ferramentas que possibilitem automatizar tarefas e simplificar etapas do projeto desses sistemas tem sido crescente. Estas ferramentas de projeto objetivam fornecer ao projetista uma "interface inteligente" visando absorver as especificações funcionais e não funcionais do sistema em níveis elevados de abstração. As técnicas de inteligência artificial [16], mais especificamente os métodos de aprendizado automático, têm desempenhado um papel importante nesse cenário.

Uma das características dos métodos de inteligência artificial é necessitar de um computador para executar o algoritmo ou a heurística de busca de soluções. De fato, a inteligência artificial é calcada, fortemente, em experimentos. No entanto, observa-se que a maioria dos métodos de aprendizado automático hoje disponíveis, como métodos de indução, sistemas auto-organizáveis, redes neurais e algoritmos genéticos, não busca soluções na forma de programas, isto é, representações de procedimento. Ao contrário, estes paradigmas de aprendizado utilizam estruturas especializadas bastante diferentes de programas de computador. Por exemplo, enquanto as redes neurais utilizam vetores de peso, os algoritmos genéticos normalmente trabalham com cadeias de bits de tamanho fixo. Estas estruturas de representação podem, por vezes, facilitar soluções para de-

terminadas classes de problemas ou ser mais adequadas a uma análise matemática, porém todas elas dificultam ou restringem a forma de fazer com que computadores solucionem seus problemas diretamente.

Diferentemente dos demais paradigmas de aprendizado automático, a programação genética enfatiza características desejáveis, tal como o uso de representações de procedimentos, gerando soluções diretamente como programas de computador. Como conseqüência, e desde que os problemas que se deseja resolver possam ser reformulados como problemas de indução de programas, ela se torna uma ferramenta mais adequada para a solução desses problemas.

A seção 2.2 apresenta os fundamentos de inteligência artificial e os principais paradigmas de aprendizado automático. A seção 2.3 detalha a programação genética, apresentando em seguida uma revisão literária de abordagens de programação genética orientada à gramática.

2.2 Inteligência Artificial: Sistemas de Aprendizado Automático

Inteligência artificial (IA) é a ciência computacional que se interessa pelo estudo e criação de sistemas que possam exibir comportamento inteligente [16]. Embora a IA possa explorar um espaço de possíveis "inteligências", a inteligência é geralmente entendida como aquela expressa por seres humanos no que se refere à capacidade de interagir com um ambiente parcialmente conhecido e que se altera em função do tempo. Pesquisadores de IA sustentam que tarefas complexas, com nível de competência equivalente ou superior ao de um especialista humano, podem ser realizadas por computadores.

Apesar de ligada a várias teorias, IA é considerada uma ciência empírica [16]. Seu foco de estudo são problemas que geralmente não respondem a soluções algorítmicas e dos quais não se dispõe de informação completa. Os dois principais fundamentos da IA são:

- Representação do conhecimento; e
- Busca, ou seja, exploração do espaço de estados (ou soluções) do problema.

A função de qualquer forma de representação é capturar os aspectos essenciais de um (domínio de) problema e tornar tais informações acessíveis aos procedimentos que irão solucioná-lo. Na construção de uma base de conhecimento, o programador deve selecionar objetos (símbolos) e relações em um domínio de problema e mapeá-los em uma linguagem formal.

Linguagens de representação de conhecimento são ferramentas importantes para auxiliar pessoas na resolução de problemas. Algumas delas são utilizadas quase que exclusivamente em aplicações de IA. LISP (*LISt Processing*) e PROLOG (*PROgramming LOGic*) são os exemplos mais comuns. Enquanto LISP é uma linguagem de procedimentos fundamentada na teoria de recursão, PROLOG é basicamente uma máquina de inferência para o cálculo de predicado de primeira ordem.

Por outro lado, problemas são resolvidos pela busca entre as alternativas presentes em um espaço de soluções. O espaço de soluções é o conjunto de todas as soluções que podem ser encontradas pelo algoritmo ou heurística de busca, dada uma forma de representação. Há várias técnicas para realizar esta busca. Todas elas utilizam um processo adaptativo ou inteligente no qual a informação obtida em um estado da busca é utilizada para influenciar a direção futura da busca [5].

No aprendizado automático, ou aprendizado de máquina (AM), investiga-se os mecanismos pelos quais o conhecimento é adquirido através de experiências, conferindo aplicação prática à IA. A qualidade e a quantidade de exemplos, isto é, as informações de treinamento, são fundamentais para qualquer algoritmo de aprendizado. Aprendizado envolve mudança no aprendiz, podendo ser definido como "*qualquer modificação no sistema que o permita efetuar melhor na segunda vez pela repetição da mesma tarefa ou em outra tarefa amostrada da mesma população*" [17]. Há basicamente três abordagens para AM [16]:

- Baseada em símbolos (*symbol-based*);

- Redes neurais (*connectionist*); e
- Emergente e social.

Newell introduziu seu sistema de símbolos físicos (*physical symbol system*) [18] como uma classe de sistemas que corporifica a natureza essencial dos símbolos: "O *estudo de lógica e computadores tem nos revelado que a inteligência reside em um sistema de símbolos físicos. Sistemas de símbolos são coleções de padrões e processos, sendo os últimos capazes de criar, destruir e modificar os primeiros...*" [19]. Essa teoria distingue *padrões* formados por *arranjos de símbolos* do *meio* utilizado para implementá-los, fazendo entender que qualquer meio que implemente corretamente esses padrões e processos seria assumido como "agente inteligente", seja ele composto por neurônios ou circuitos lógicos.

Em modelos baseados em símbolos, o conhecimento é representado explicitamente: a informação é codificada em um sistema de símbolos e o aprendizado se dá através da manipulação de estruturas de símbolos. Baseado na sua experiência, o modelo constrói ou modifica expressões (símbolos) em uma linguagem formal, como por exemplo, LISP ou PROLOG, e retém este conhecimento para uso futuro. Aprendizado baseado em símbolos inclui indução, que pode ser definida como "*aprendendo uma generalização de um conjunto de exemplos*" [16].

Modelos inspirados no neurônio biológico, conhecidos como PDP (*Parallel Distributed Processing*) ou *conexionista*, retiram a ênfase no uso explícito de símbolos ao resolverem problemas. Ao invés disso, consideram que a inteligência é suportada por sistemas de componentes simples (neurônio biológico ou artificial) que se interagem através de um processo de aprendizado ou adaptação pelo qual as conexões entre estes componentes são ajustadas.

Nas redes neurais, o conhecimento está implícito na organização e interação desses neurônios. O processamento é distribuído através de coleções de camadas de neurônios. O mecanismo de solução do problema é paralelo no sentido de que todos os neurônios pertencentes a uma mesma camada processam suas entradas simultaneamente e independentemente. As redes neurais funcionam melhor em tarefas nas quais os modelos baseados em símbolos apresentam baixo rendimento, isto é, quando há pouca informa-

ção sobre o problema ou quando há presença de ruído nos dados de entrada.

Assim como as redes neurais, os modelos emergentes são baseados em uma analogia biológica. Nesses modelos, o aprendizado é tido como um processo emergente e adaptativo semelhante a processos de evolução encontrados em seres humanos ou animais: a evolução se dá através de gerações, pela sobrevivência dos membros mais aptos da população. Modelos sociais (*society-based learning*) são uma forma específica de modelos emergentes. Nesses modelos, processos sociais, tais como alterações culturais conduzidas pela modificação e transmissão de unidades de informação, exercem poder de seleção sobre uma população de indivíduos.

Devido à multiplicidade de estudos dedicados a IA, os pesquisadores estão construindo uma hierarquia de teorias que caracterizam-na em vários níveis de abstração. No nível mais baixo dessa hierarquia estão as redes neurais e os modelos emergentes. Nesses modelos, populações de agentes primitivos se interagem e se adaptam para gerar conhecimento. A abordagem nesse nível é totalmente empírica. Em um nível intermediário estão as abordagens que utilizam indução ou outras formas de raciocínio lógico. No nível mais alto dessa hierarquia podem ser citados os sistemas especialistas, isto é, sistemas baseados em símbolos que utilizam alta densidade de informação sobre um assunto específico e processos de inferência que incluem aspectos sociais para realizar uma tarefa específica.

Computação Evolutiva

Algoritmos probabilísticos inspirados em princípios da seleção natural e suas variações beneficiam-se do aumento do poder de simulação para resolver problemas complexos, problemas de controle e aplicações de aquisição de conhecimento. Esses algoritmos são objeto de um campo emergente de evolução simulada denominado *computação evolutiva* ou *evolucionária* [4]. Exemplos de computação evolutiva são: (a) programação evolutiva [20, 21]; (b) algoritmos genéticos [22, 23]; (c) estratégias evolutivas ou de evolução [24, 25]; (d) sistemas classificadores [26]; e (e) programação genética [5].

Os algoritmos evolutivos implementam as diversas instâncias da computação evolutiva. Tais algoritmos são estratégias gerais de busca, sendo, portanto, classificados como

métodos fracos de solução de problemas. Estes métodos partem de requisitos relaxados, requerendo pouco ou nenhum conhecimento do domínio do problema, tendo, conseqüentemente, um amplo leque de aplicação. O problema dos métodos fracos é que o mecanismo de controle de busca pode não ser suficientemente poderoso. Outra desvantagem desse método, conseqüência da falta de restrições, é que ele é fonte de representações irregulares. Em contraste, um *método forte* de solução de problemas é aquele que se aprovisiona de uma quantidade significativa de conhecimento específico do domínio do problema que pretende resolver. O problema dos métodos fortes, como por exemplo, dos sistemas especialistas, é que o conhecimento explícito torna-se um produto que determina a eficácia da técnica.

Pesquisadores que sustentam a teoria da computação evolutiva, ou emergente, questionam se os sistemas baseados em símbolos exibem verdadeira inteligência. Sistemas de IA simbólica são projetados ou programados, ao invés de serem treinados ou evoluídos. Esses sistemas são tipicamente confinados a tarefas específicas, sendo raramente efetivos fora de seu domínio.

Algoritmo Genético

O algoritmo genético (AG), a forma mais popular de computação evolutiva, foi introduzido formalmente por Holland [22] como forma de aprendizado adaptativo. O AG é baseado no princípio da seleção natural de Darwin e nos mecanismos da genética, onde uma população de indivíduos é moldada através de gerações pela sobrevivência dos mais aptos.

AGs são freqüentemente descritos como métodos de busca global que não utilizam a informação do gradiente. Assim sendo, funções não diferenciáveis e funções multimodais, isto é, aquelas com múltiplos mínimos locais, representam problemas para os quais o AG pode ser aplicado. Por essa razão, o AG tem sido empregado em tarefas de otimização altamente complexas e multidimensionais, onde outras estratégias normalmente fracassam. Em particular, os AGs funcionam bem em problemas de natureza combinatória, de variáveis discretas ou contínuas.

O AG se inicia com uma população de indivíduos criados com distribuição unifor-

me, a geração zero. Operações (reprodução, cruzamento, mutação, cálculo de aptidão e seleção) são executadas sobre estes indivíduos (cromossomos codificados) e uma nova população é gerada. A aptidão é uma medida utilizada para indicar quão próximo um indivíduo está da solução. Na escolha (seleção) dos indivíduos participantes da nova população é sempre levado em conta o grau de aptidão do indivíduo. Assim, os indivíduos mais aptos têm mais chance de permanecer na nova população.

Nesse processo, cada indivíduo é um ponto no espaço de soluções do problema. O estado (memória) do sistema é, portanto, uma população de soluções. O AG, assim como a PG, é considerado uma forma de *busca em feixe* (*beam search*), pois retém uma população de soluções potenciais que é menor que o conjunto de todas as possíveis soluções [3]. Por conseqüência, os AGs são menos susceptíveis a mínimos locais que métodos de busca que utilizam gradiente. Em compensação, exigem maior esforço computacional por manipularem populações inteiras de possíveis soluções.

O teorema dos esquemas (*schema theorem*), introduzido em [22], foi utilizado como tentativa de formalizar o funcionamento dos AGs. Segundo este teorema, operadores genéticos manipulam os *esquemas*, isto é, *strings* de bits que funcionam como *blocos construtores* (*building blocks*) ou partes fundamentais, que são utilizados na geração de famílias de soluções em direção a soluções potenciais.

Algumas melhorias foram introduzidas no AG clássico ou simples, apresentado em [22], com o objetivo de: (a) manter a diversidade da população, por exemplo, através de métodos de compartilhamento de recursos [27]; (b) aumentar sua capacidade de busca utilizando, conjuntamente ao AG, outros paradigmas de busca, o que é conhecido como hibridização [28]; e (c) tratar mais de um objetivo na função de otimização (vide [29]).

2.3 Programação Genética

A programação genética (PG) (GP: *Genetic Programming*) é derivada do AG clássico, descrito em [22]. O termo *Genetic Programming* foi cunhado independentemente por John Koza e Hugo DeGaris através de trabalhos publicados no início da década de 90 [3]. A principal diferença entre o AG e a PG está na representação da solução. Ao

contrário do AG, a PG utiliza genótipos de tamanhos variados o que lhe confere maior potencial na criação de novas estruturas.

A solução de problemas com uso de PG requer inicialmente a definição de uma linguagem de representação para criar soluções alternativas e então prover uma medida de aptidão. A PG codifica soluções alternativas como programas (expressões de processos), fazendo uso de uma linguagem formal.

Como PG é um programa, e já que outros sistemas de AM (diferentes de PG) são ou podem ser executados em computadores, ou seja, podem ser representados por programas de computador, PG é capaz, teoricamente, de evoluir qualquer solução que possa ser encontrada por quaisquer dos demais sistemas de AM. Dessa forma, a representação de problemas em PG é considerada um *superconjunto* de todas as outras representações de AM [3].

A referência [5] não apresenta prova matemática de que PG possa ser sempre bem sucedida na solução de problemas. Ocorre que, na natureza, a diferença entre a estrutura hoje observada e sua antecessora não é explicada através de provas matemáticas ou coisas do gênero. Ao contrário, Koza apresenta evidências empíricas, através de vários exemplos em diversos domínios de aplicação, de que PG produz resultados satisfatórios em um grande número de problemas aparentemente diferentes. Koza aborda dois pontos: (a) a viabilidade de reformulação da grande maioria dos problemas como problemas de indução de programas; e (b) a indução de programas por PG, isto é, PG promove a descoberta de programas de computador que produzem as saídas desejadas quando apresentados a entradas particulares.

O conjunto de possíveis estruturas em PG é o conjunto de todas as possíveis composições de funções e de terminais. A escolha de funções e de terminais é feita pelo usuário e deve atender a *propriedade de suficiência* (*sufficiency property*). Especificamente para PG, essa propriedade estabelece que o conjunto de funções (operações) e o conjunto de terminais (variáveis e constantes) devem ser capazes de expressar a solução do problema. Portanto, o usuário deve saber *a priori* que a composição de funções e terminais por ele fornecida induz à solução do problema.

Por outro lado, a *propriedade de fechamento* (*closure property*) requer que cada função, pertencente ao conjunto de funções, seja capaz de aceitar, como argumento,

qualquer valor e tipo de dado que possa ser retornado por outras funções (pertencentes a este conjunto) e qualquer valor e tipo de dado que possa ser assumido por um terminal pertencente ao conjunto de terminais [5]. O atendimento a esta propriedade permite que o cruzamento efetuado entre dois nós quaisquer da árvore de derivação produza indivíduos sintaticamente corretos. Suficiência e fechamento são as principais restrições impostas às operações de cruzamento e mutação na PG canônica [30].

A figura 2.1 mostra o fluxograma de execução de PG. Nessa figura, *i* representa um indivíduo, *Gen* representa o número da geração corrente e *M* representa o tamanho da população. A PG emprega os seguintes passos para resolver problemas:

1) Geração de população inicial de composições aleatórias de funções e terminais definidos para o problema (programas de computador).
2) Realização iterativa dos seguintes passos até que o critério de terminação seja alcançado:
   a) Execução de cada programa da população e estabelecimento de um *valor de aptidão* de acordo com sua potencialidade em resolver o problema.
   b) Criação de uma nova população de programas pela aplicação das seguintes operações em indivíduos escolhidos com probabilidade baseada na aptidão:
      b.1) Cópia de programas existentes para a nova população através do operador de *reprodução* (reprodução assexual ou reprodução darwiniana);
      b.2) Criação de novos programas através do operador de *cruzamento* (reprodução sexual ou recombinação);
      b.3) Criação de novos programas através do operador de *mutação*.
3) Escolha do melhor dentre os programas de todas as gerações, assumindo este como a solução do problema.

É importante observar que a PG não garante o aparecimento do melhor indivíduo em todas as gerações subseqüentes, exceto quando se emprega a estratégia do *elitismo*. Portanto, na designação do resultado deve-se rastrear o melhor indivíduo dentre todas as gerações, o que é feito armazenando em separado o melhor indivíduo até então gerado (*best-so-far*).

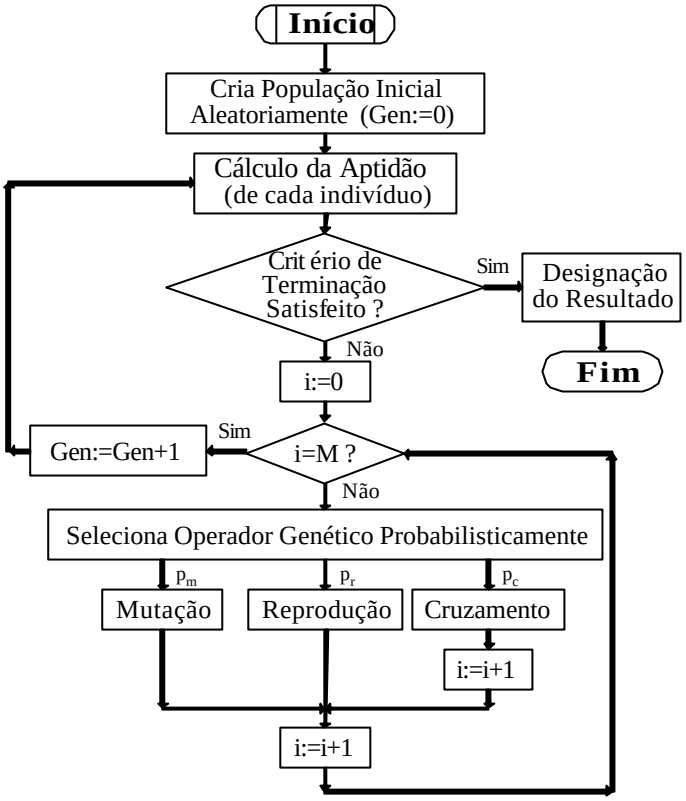


Figura 2.1: Fluxograma de execução de PG.

O critério de terminação adotado por Koza estabelece que o algoritmo termina quando: (a) um número máximo de gerações foi executado (*predicado de geração*); ou (b) algum atributo de sucesso específico ao problema foi satisfeito (*predicado de sucesso*). O predicado de sucesso geralmente envolve a captura de um indivíduo 100% apto. Quando tal indivíduo não é encontrado utiliza-se um critério para sucesso inferior a 100%, adequado ao problema em questão.

Koza utilizou a linguagem LISP em seus experimentos. Uma importante característica de LISP é que todo programa nele realizado tem somente uma forma sintática, as expressões simbólicas (*S-expressions*), não se distinguindo entre dado e programa. Como efeito, é possível tratar um programa na população genética como dado de maneira que ele possa ser manipulado geneticamente. Em seqüência, é possível e conveniente executar o resultado de uma manipulação genética como programa. A figura 2.2 apresenta o resultado da operação de cruzamento entre dois programas em LISP, gerando dois descendentes pela recombinação de material genético a partir da definição de pontos de cruzamento nos pais (ascendentes). É importante notar que programas em LISP

são equivalentes às suas árvores sintáticas.

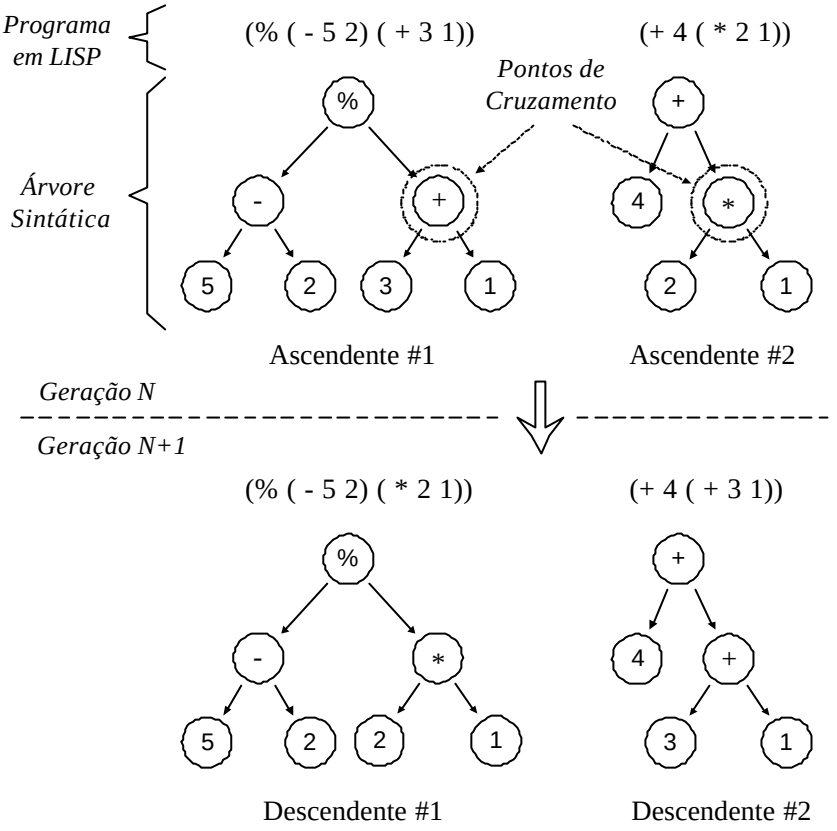


Figura 2.2: Operação de cruzamento entre programas em LISP.

A introdução formal de PG [5] apresentou uma forma de evolução de programas em LISP pela manipulação de estruturas de árvores (genomas do tipo árvore). A partir de então, outros tipos de genoma foram desenvolvidos por outros pesquisadores, dentre os quais destacam-se os genomas lineares e os genomas baseados em grafos.

Genomas lineares são cadeias de bits que traduzem melhor a estrutura fundamental da genética biológica, o DNA (*Deoxyribonucleic Acid*). Esta estrutura é constituída por uma cadeia de moléculas lineares encontradas em todas as células vivas, contendo a informação genética. O fenótipo linear é simplesmente uma lista de instruções a serem executadas da esquerda para direita, ou de cima para baixo. AIMGP (*Automatic Induction of Machine code with Genetic Programming*) [31] é um exemplo de sistema deste tipo. Tal sistema representa programas como seqüências lineares de instruções em código de máquina que são diretamente interpretadas pela CPU.

Genomas baseados em grafos são mais complexos e menos explorados. Não obstante, seu campo de aplicação é potencialmente elevado em problemas que podem ser descritos por meio de grafos, como por exemplo, circuitos, redes neurais e máquinas de estados finitos [3]. Grafos são superconjuntos de árvores e de listas. A estrutura de um grafo é composta por nós e arestas, que conectam dois nós. Uma árvore é um tipo especial de grafo acíclico com um nó especial, o nó raiz. Uma lista (genoma linear) é um tipo especial de árvore em que cada nó está ligado a, no máximo, dois nós vizinhos. *Developmental System* [32] e PADO (*Parallel Algorithm Discovery and Orchestration*) [33] são exemplos de sistemas com genomas baseados em grafos.

Outras estruturas de genoma têm sido propostas, muitas das vezes utilizando como base os genomas acima mencionados. Como exemplo, o genoma proposto no sistema STROGANOFF [34] utiliza uma estrutura polinomial dentro dos nós de uma árvore de derivação para modelar sistemas a partir de amostras de dados. Outros sistemas [35, 36] utilizam gramática livre de contexto como estrutura de evolução para contornar o requisito de fechamento. Tais sistemas, conhecidos como PG orientada à gramática, são de interesse particular deste trabalho e serão detalhados na subseção 2.3.3.

Independentemente do tipo de genoma utilizado no processo evolutivo, as idéias computacionais de PG podem ser resumidas como se segue [4]:

- Uso de representações de processos;
- Busca entre alternativas paralelas através de uma população de representações de processos;
- Simulação, ou seja, execução da solução candidata em meio a um ambiente para conferir um valor de retorno ou aptidão e classificação entre alternativas de acordo com este valor;
- Competição entre alternativas baseada em seleção estocástica das melhores alternativas;
- Variação estocástica de alternativas selecionadas para diversificar a busca.

2.3.1 Parâmetros de Controle de PG

Os parâmetros de controle de PG totalizam 19 e podem ser distribuídos em três grupos [5]: (a) parâmetros numéricos principais (*major numerical parameters*), determinantes no desempenho geral do sistema; (b) parâmetros numéricos secundários (*minor numerical parameters*), utilizados para controlar o processo de evolução; e (c) variáveis qualitativas, que selecionam entre diferentes formas de execução do sistema. Estes parâmetros são relacionados a seguir com os respectivos símbolos, quando aplicáveis, e valores ou condição *default*, entre parênteses, para os problemas clássicos de PG apresentados em [5]:

Parâmetros Numéricos Principais:

- **M**: tamanho da população (M = 500);
- **G**: número máximo de gerações em uma única execução (G = 51).

Parâmetros Numéricos Secundários:

- $\mathbf{p_c}$: probabilidade de cruzamento (p_c = 0,90);
- $\mathbf{p_r}$: probabilidade de reprodução (cópia ou clonagem) (p_r = 0,10);
- $\mathbf{p_m}$: probabilidade de mutação (p_m = 0,0);
- $\mathbf{p_p}$: probabilidade de permutação (p_p = 0,0);
- $\mathbf{p_{ip}}$: distribuição de probabilidade de cruzamento (p_{ip} = 90%);
- $\mathbf{f_{ed}}$: freqüência de edição (f_{ed} = 0);
- $\mathbf{p_{en}}$: probabilidade de encapsulamento (p_{en} = 0,0);
- $\mathbf{D_{created}}$: máxima profundidade da árvore após um cruzamento ($D_{created}$ = 17);
- $\mathbf{D_{initial}}$: máxima profundidade da árvore da população inicial ($D_{initial}$ = 6);
- Condição para invocar decimação (nula);
- $\mathbf{p_d}$: porcentagem de decimação (p_d = 0,0).

Variáveis Qualitativas:

- Método de geração da população inicial aleatória (*ramped half-and-half*);

- Método de seleção para reprodução e de seleção para o primeiro pai no cruzamento (proporcional à aptidão);
- Método de seleção para o segundo pai no cruzamento (proporcional à aptidão);
- Utilização ou não de ajuste de aptidão (*adjusted fitness*) (utilizado);
- Utilização ou não de superseleção (*over-selection*) (não utilizado quando $M \geq 500$; utilizado quando $M < 500$);
- Utilização ou não da estratégia de elitismo (não utilizada).

A exclusão do operador de mutação em sistemas de PG tem sido contestada por diversos trabalhos de pesquisa subseqüentes à referência [5], tais como [3, 37], principalmente pela incapacidade desses sistemas em preservar a diversidade genética da população.

O tamanho da população M é o parâmetro mais importante de PG [5] e deve ser escolhido de acordo com a complexidade do problema a ser resolvido. Como regra geral, o aumento da população reforça o poder de busca do algoritmo. De fato, uma população pode ser entendida como tendo uma "capacidade de conhecimento" que se refere à quantidade de informação que ela mantém disponível. No entanto, deve-se observar que a melhora devido ao aumento da população pode não ser proporcional ao aumento de recursos computacionais demandados. O valor de M mais utilizado em [5] é 500. Variações deste valor são encontradas a partir de 50, valor utilizado para funções booleanas de dois ou três argumentos, podendo chegar a valores superiores a 1000000, valor utilizado utilizado em experimentos no sistema AIMGP [3]. A população, na maioria das implementações, permanece constante no decorrer da evolução, embora não haja, *a priori*, nenhuma razão específica para este fato, salvo a conveniência de se fazer essa suposição [1].

Na primeira metade dos anos 90, o número máximo de gerações G, outro importante parâmetro de PG, vinha sendo estabelecido em 51 com o argumento de que nada acontecia após esta geração. No entanto, experiências recentes revelaram evoluções interessantes em gerações superiores a este limite. Em [3] é proposto um método simples para se estabelecer G: inicia-se o teste com $50 \leq G \leq 100$; caso não resulte na evolução desejada deve-se, primeiramente, aumentar M e, posteriormente, G.

O número de execuções do sistema de PG é outro fator que pode ser determinante na busca por soluções de problemas. O aumento do número de execuções independentes minimiza o aparecimento de nichos (sub-populações com espécies dominantes), a convergência prematura do algoritmo e o efeito de condições iniciais na busca por soluções. Portanto, deve-se optar por rodar o sistema mais vezes ao invés de executá-lo menos freqüentemente com *G* maior.

2.3.2 Quantidade de Processamento de PG

Um dos métodos utilizados para medir o desempenho de um sistema de PG é o cálculo da quantidade de processamento necessário para se obter uma solução com uma determinada probabilidade de sucesso, geralmente 99%. Uma vez que PG envolve probabilidade, por exemplo, na criação da população inicial ou na escolha de pares para o cruzamento, não há garantia de convergência ou resposta a um problema em uma determinada execução. Portanto, essas medidas devem ser apresentadas mediante probabilidade.

A quantidade de processamento necessária para se obter uma solução com uma determinada probabilidade de sucesso é diretamente proporcional ao número total de indivíduos processados (tamanho da população × número de gerações × número de execuções) e fornece uma medida da complexidade do problema. A quantidade de processamento é influenciada também pelo processamento demandado para se medir a aptidão de um indivíduo frente a todos os casos de treinamento aplicáveis.

Para realizar o cálculo da quantidade de processamento é necessário primeiro obter experimentalmente a Probabilidade Instantânea *Y(M,i)* de que uma certa execução com *M* indivíduos produza pela primeira vez, exatamente na geração *i*, uma solução que satisfaça ao predicado de sucesso. Esta medida pode ser obtida experimentalmente realizando-se várias execuções independentes com os mesmos parâmetros *M* e *i*.

Uma vez determinada *Y(M,i)* para todo *i* (todas as gerações), a Probabilidade de Sucesso *P(M,i)* para a geração até *i* pode ser computada. *P(M,i)* é uma medida cumulativa de sucesso, fornecendo a probabilidade de se obter uma solução para um dado problema

quando é realizada uma execução com i gerações, isto é, $P(M,i) = Y(M,0) + Y(M,1) + ... + Y(M,i)$.

Após algumas manipulações aritméticas, chega-se à probabilidade z de se encontrar uma solução pela geração i, utilizando R execuções, ou seja, $z = 1-(1-P(M,i))^R$ [5]. Mais interessante será conhecer a quantidade de execuções necessárias para resolver o problema com probabilidade z, o que é dado por $R = \log(1-z)/\log(1-P(M,i))$, onde R deve ser aproximado para o próximo inteiro superior.

Em seguida calcula-se $I(M,i,z)$, isto é, o número total de indivíduos que devem ser processados para produzir o resultado desejado pela geração i, com probabilidade z, utilizando uma população de tamanho M. Para tal, multiplica-se o tamanho da população M pelo número de gerações i e pelo número de execuções R. Finalmente, a *quantidade de processamento* ou *esforço computacional* é obtido multiplicando-se $I(M,i,z)$ pela quantidade de processamento necessário para medir a aptidão de cada indivíduo sobre todos os casos de treinamento aplicáveis. Como o valor da medida de aptidão varia entre indivíduos, utiliza-se um valor médio.

As curvas de desempenho apresentadas em [5] são mostradas com $P(M,i)$ e $I(M,i,z)$ sobrepostas no eixo vertical em função do número de gerações apresentadas no eixo horizontal. Resumidamente, as etapas para a obtenção destas curvas são:

- Realização de várias execuções do sistema para se obter $Y(M,i)$;
- Cálculo de $P(M,i)$ por geração fazendo-se $P(M,i) = Y(M,0) + Y(M,1) + ... + Y(M,i)$: obtém-se, então, a curva $P(M,i)$ em função do número de gerações;
- Determinação do número de execuções independentes R necessárias para produzir, pelo menos, uma execução com 99% de probabilidade de sucesso utilizando-se $R = \log(1-z)/ \log(1-P(M,i))$;
- Cálculo de $I(M,i,z)$, fazendo-se $M \times i \times R$;
- Apresentação do gráfico com G no eixo horizontal, $P(M,i)$ no eixo vertical esquerdo e $I(M,i,z)$ no eixo vertical direito.

2.3.3 PG Orientada à Gramática

A PG convencional introduzida por Koza baseou-se inteiramente em experimentos utilizando LISP como linguagem de programação. Um dos principais motivos para a escolha de LISP é que sua expressão simbólica (isto é, seu programa) é equivalente à árvore de derivação ou árvore sintática (*parse tree*) (Cf. figura 2.2). O acesso à árvore sintática se faz necessário para manipular geneticamente partes de programas.

Na maioria das linguagens de programação, a árvore sintática não é acessível, ou não é convenientemente acessível, apesar de o processo de compilação destas linguagens gerar, na maior parte das vezes, uma árvore sintática representando a composição de funções e terminais do programa [5]. Dessa forma, o sistema de PG proposto por Koza mostra-se inadequado para gerar programas em linguagens com estruturas gramaticais diferentes das utilizadas em LISP.

A utilização de sistemas que permitam gerar automaticamente programas em linguagens arbitrárias é desejável pela conveniência de se escolher uma linguagem de programação específica para um determinado domínio de aplicação. No período que se seguiu à introdução de PG, vários sistemas foram desenvolvidos com este propósito [35, 36, 38, 39, 40, 41]. A maioria deles utiliza como entrada uma BNF (*Backus-Naur Form*), notação formal freqüentemente utilizada para especificar a sintaxe de linguagens de programação [42]. Uma BNF descreve estruturas gramaticais admissíveis na construção de uma linguagem através de uma 4-tupla $\{S, N, T, P\}$, onde S denota o símbolo inicial, N denota o conjunto de símbolos não terminais, T denota o conjunto de símbolos terminais e P são as produções. Uma produção, ou regra de transformação de *string*, é uma substituição do tipo $X \rightarrow Y$, onde $X \in N$ e $Y \in (N \cup T)^*$. As produções a seguir, adaptadas de [43], possibilitam gerar uma função contendo várias linhas de código em uma linguagem hipotética:

```
S                ::=  <func>;
<func>           ::=  <header>;
<header>         ::=  "float myfunction(float x)" <body>;
<body>           ::=  <declarations> <code> <return>;
<declarations>   ::=  "float a;";
<return>         ::=  "return(a);";
<code>           ::=  <line> | <code> <line>;
```

```
<line>          ::=   <expr>;
<expr>          ::=   <expr> <op> <expr> | <pre-op> "(" <expr> ")" |
                      <var>;
<op>            ::=   "+" | "-" | "/" | "*";
<pre-op>        ::=   "sin" | "cos" | "tan" | "log";
<var>           ::=   "x";
```

No exemplo acima, *S* é o símbolo inicial, símbolos entre chaves são símbolos não terminais e símbolos entre aspas são símbolos terminais. O sinal "::=" denota "é substituído por". Maneiras alternativas de reescrever um determinado símbolo não terminal são separadas por uma barra vertical. Uma linguagem $L(G)$ gerada por uma gramática G, definida por uma BNF, é o conjunto de todas sentenças que podem ser derivadas a partir de *S*. Para se criar uma sentença em uma linguagem particular é necessário escolher quais produções serão utilizadas e em que seqüência elas serão utilizadas, função esta destinada ao algoritmo evolutivo. Os seguintes sistemas de PG orientada à gramática foram encontrados na literatura e são descritos sucintamente a seguir:

- **LOGENPRO** (*LOgic grammar-based GENetic PROgramming*);
- **CFG-GP** (*Context-Free Grammar-Genetic Programming*);
- **GPK** (*Genetic Programming Kernel*);
- **GPM** (*Genotype-Phenotype Mapping*);
- **GADS** (*Genetic Algorithm for Deriving Software*);
- **GE** (*Grammatical Evolution*).

LOGENPRO [39]:

Uma das principais contribuições dessa abordagem é demonstrar que uma gramática lógica pode ser utilizada para combinar PG com ILP (*Inductive Logic Programming*), possibilitando a construção de um sistema capaz de representar informação sensível a contexto. LOGENPRO (*LOgic grammar-based GENetic PROgramming*) utiliza genomas do tipo árvore e, como em [35], os operadores de cruzamento e mutação são aplicados em árvores de derivação. No entanto, este sistema sofre de ambigüidades por não manter explícitas, na população, árvores com estrutura sintática [41, 44]. LOGENPRO foi utilizado com sucesso para emular fun-

ções definidas automaticamente (ADFs: *Automatically Defined Functions* [45]) com objetivo de descobrir soluções para o problema de representação de funções primitivas.

CFG_GP [30, 35, 44]:

Uma maneira elegante e promissora de codificar o conhecimento em um certo domínio de problema é lançar mão de gramáticas formais. CFG-GP (*Context-Free Grammar-Genetic Programming*) introduz uma forma geral de PG baseada em gramáticas formais, conhecida como G^3P (*Grammar-Guided Genetic Programming*) [46]. Restrições em produções (regras de re-escrita) determinam classes na hierarquia de linguagens formais apresentada por Chomsky [47]. Gramáticas livres de contexto (CFG: *Context-Free Grammar*) ocupam o terceiro nível dessa hierarquia e caracterizam-se por possuir um único símbolo não terminal no lado esquerdo das produções. Na construção de uma sentença de programa, as produções são aplicadas até que todos os símbolos não terminais sejam substituídos por símbolos terminais. A notação BNF é equivalente à gramática livre de contexto no sentido de que seus símbolos expandem-se sem fazer referência ao contexto adjacente [42].

O requisito de fechamento postulado em [5] dificulta a expressão de muitas estruturas de programas [35]. CFG-GP utiliza gramática livre de contexto para garantir a geração de programas "legais" e contornar este requisito. Em [30] definem-se restrições quanto à linguagem (*language bias*), aplicáveis a sistemas de aprendizado evolutivo. Tais restrições, que dizem respeito ao modo com que possíveis estruturas são criadas, podem ser descritas de duas formas:

- Restrição declarativa (*declarative bias*): define formas possíveis de estruturas de programas iniciais e subseqüentes (descrição BNF fixa);
- Restrição instruída (*learnt bias*): modifica, durante o processo evolutivo, o modo pelo qual formas possíveis de sentenças são geradas e introduzidas na população (descrição BNF induzida). Essa restrição permite ao sistema guiar a busca em direção a partes específicas da gramática melhorando a convergência do algoritmo

[41]. Adicionalmente, possibilita a criação de blocos construtores agrupando várias funções em uma função única.

GPK [36]:

GPK (*Genetic Programming Kernel*) utiliza árvores de derivação para gerar programas em linguagem arbitrária. Apesar de ter sido desenvolvido isoladamente, seu sistema é bastante semelhante a CFG-GP já que a linguagem, ou parte dela, é especificada através de uma descrição BNF. Assim sendo, nenhum reparo é necessário no processo de criação de uma sentença, uma vez que todos indivíduos gerados são válidos. No entanto, GPK é criticado em [40] pela dificuldade em gerar a população inicial. Como conseqüência, sistemas GPK que funcionam com o operador de mutação anulado ficam com o desempenho comprometido como conseqüência da falta de diversidade genética. GPK é um sistema complexo e foi adotado como núcleo do sistema de programação genética utilizado neste trabalho. Essa escolha deveu-se a critérios de flexibilidade e portabilidade. O sistema GPK é um sistema inteiramente parametrizado e pode ser facilmente compilado para execução no sistema operacional Windows, por exemplo. Este sistema implementa os três principais tipos de cálculo de aptidão (aptidão padronizada, ajustada e normalizada), assim como os principais métodos de seleção. O número de pontos de cruzamento pode ser livremente estabelecido. GPK permite adicionar indivíduos específicos à população inicial, técnica conhecida como *coaching*. Entretanto, seu procedimento de inicialização foi alterado para contornar a dificuldade mencionada anteriormente.

GPM [37, 38]:

Enfoques comuns de PG convencional defrontam-se com uma forte restrição oriunda da sintaxe de linguagens de programação na qual todos os indivíduos gerados devem ser “legais”, isto é, sintaticamente corretos. Neste contexto, grandes regiões do espaço de busca tornam-se inviáveis, o que o restringe sobremaneira. Na PG convencional não há distinção entre genótipo, um ponto no espaço de busca, e fenó-

tipo, um ponto no espaço de soluções, fazendo entender que o espaço de busca e o espaço de soluções sejam idênticos. GPM (*Genotype-Phenotype Mapping*), no entanto, distingue o espaço de busca do espaço de soluções. A justificativa para tal distinção é baseada no fato de que a evolução molecular é impulsionada essencialmente por *mutações quase neutras* em relação à seleção natural. Tal comportamento baseia-se na teoria neutra da evolução molecular, postulada em [48] e suportada por resultados empíricos apresentados em [49]. Este fenômeno é o maior responsável pela alta diversidade genética em populações naturais: por exemplo, na população de seres humanos não há duas pessoas idênticas. Não descartando indivíduos "ilegais", e mapeando-os em indivíduos legais, GPM aumenta a diversidade genética evitando que o sistema estacione em mínimos locais no espaço de soluções.

GPM utiliza genoma linear de tamanho fixo e descreve um mapeamento de *muitos* genótipos para *um* único fenótipo (*many-to-one mapping*). Em uma das fases deste mapeamento, denominada *reparação* (*repairing*), que transforma um símbolo mal evoluído (ilegal) em um símbolo legal, GPM restringe sua aplicação a gramáticas do tipo LALR(1) (*Look Ahead Left Recursive – look ahead one symbol*). Linguagens baseadas em gramáticas deste tipo, como por exemplo, a linguagem de programação 'C', apresentam uma conveniente propriedade no que se refere à compilação, já que é sempre possível computar o conjunto de símbolos legais com respeito ao último símbolo processado [37].

GPM utiliza somente o operador de mutação por ser este o principal operador de busca na teoria postulada em [48]. A experiência é realizada para dois tipos de mutação: irrestrita e acoplada. A mutação acoplada, quando são alterados dois bits por vez, foi empregada na experiência devido a uma analogia com o processo de mutação natural na qual nucleotídeos alteram-se aos pares.

GPM é apontado como uma melhor aproximação da mutação natural do que PG convencional, vez que somente um *códon*, que corresponde a um único símbolo, é modificado em vez da unidade sintática inteira. Na experiência apresentada em [37], em que apenas o operador de mutação foi utilizado, GPM superou PG convencional em um problema de regressão simbólica.

GADS [40]:

GADS (*Genetic Algorithm for Deriving Software*) utiliza um AG para gerar programas em linguagem arbitrária. Este sistema foi aplicado em uma série de experimentos para gerar algoritmos de *cache* utilizando um subconjunto da linguagem 'C'. Em GADS, como em GPM, genótipo e fenótipo são distintos. GADS utiliza o mecanismo do AG clássico: cromossomos de tamanho fixo (500 genes no experimento) codificam uma seqüência de produções que deve ser aplicada na formação da sentença. Se um gene, ao ser aplicado, não produz sentença sintaticamente correta, ele é ignorado. GADS introduz uma extensão na BNF fazendo que a mesma defina valores *default* para produções: se o cromossomo termina com um símbolo não terminal, um valor *default* é inserido. No entanto, uma escolha imprópria do valor *default* pode comprometer toda a evolução [41]. Este sistema apresenta o inconveniente de gerar uma quantidade excessiva de *introns*, isto é, codificação aparentemente sem função.

GE [41]:

GE (*Grammatical Evolution*) é mais uma abordagem para gerar programas em linguagem arbitrária. Em GE, o genótipo, linear e de tamanho variável, deve ser mapeado no fenótipo ao invés de ser tratado como código executável. GE faz uso da especificação de uma linguagem em BNF para gerar programas. Assim como em GADS, um AG é utilizado para codificar as produções a serem utilizadas durante o processo evolutivo. A diferença entre os dois está na forma de escolha das produções. Ao invés de codificar explicitamente a seqüência de produções a serem aplicadas na formação da sentença, GE codifica um conjunto de números pseudo-aleatórios que são utilizados no mapeamento genótipo-fenótipo para decidir sobre tal seqüência. Seu cromossomo consiste em um número variável de genes binários, cada um dos quais codificado com 8 bits. Cada um desses genes representa um número randômico que será utilizado no processo de translação genótipo-fenótipo. Este mapeamento não é direto já que na biologia natural não há mapeamento direto entre gene e sua expressão física, no caso a proteína.

Conceitos de degenerescência de código (*code degeneracy*) e seleção no estado permanente (*steady-state selection*) são utilizados em GE de forma análoga à da biologia natural. Degenerescência de código implica em produzir o mesmo fenótipo a partir de genótipos diferentes. Seleção no estado permanente traduz a tendência em manter na população indivíduos mais aptos. GE acrescenta dois novos operadores aos já existentes em AGs: o operador de duplicação (*duplicate operator*) e o operador de poda (*prune operator*). Tais operadores possibilitam ao genoma variar de tamanho com o correr do processo evolutivo. O operador de duplicação dobra um número aleatório de genes posicionando-os no final do genoma original. Uma das vantagens do uso deste operador é o aumento da presença de blocos construtores. A poda é aplicada, mediante probabilidade, a indivíduos que não expressam (mapeiam) todos os seus genes. Dessa forma, qualquer gene não utilizado no mapeamento genótipo-fenótipo será descartado. Uma conseqüência do uso do operador de poda é a eliminação de *introns*.

GE mostrou-se mais eficaz que PG convencional ao ser aplicada ao *The Santa Fe Ant Trail* [50], problema clássico caracterizado por possuir vários mínimos locais, freqüentemente utilizado pelos pesquisadores de PG para medir o desempenho de suas abordagens.

Comparação entre Sistemas de PG Orientada à Gramática

A tabela 2.1 apresenta um resumo das principais características de sistemas desenvolvidos com o propósito de gerar programas em linguagem arbitrária, descritos nesta seção.

Tabela 2.1: Principais características de sistemas que geram programas em linguagem arbitrária.

Atributos	**Sistema**					
	LOGENPRO	CFG-GP	GPK	GPM	GADS	GE
Tipo de genoma	árvore	árvore	árvore	linear	linear	linear

Especificação BNF	sim	sim	sim	não	sim	sim
Tamanho do genoma	-	variável	variável	fixo	fixo	variável
Mapeamento	-	direto	direto	indireto	indireto	indireto

2.4 Comentários

Neste capítulo foram introduzidos os fundamentos de inteligência artificial e, em especial, de abordagens de aprendizado de máquina que empregam técnicas evolutivas. A programação genética foi detalhada, sendo apresentados e discutidos seus principais parâmetros, assim como questões relativas a seu desempenho. Por fim, fez-se uma descrição literária de abordagens de programação genética orientada à gramática.

As técnicas evolutivas, assim como as redes neurais, têm sido apresentadas como poderosas opções à postura racionalista dos métodos simbólicos de aprendizado automático. Através da combinação de uma população de agentes primitivos, e da pressão adaptativa da seleção natural, os modelos emergentes, ou evolutivos, são capazes de produzir (emergir) soluções satisfatórias para o problema.

Desde a última década, as técnicas evolutivas têm transposto a barreira entre a teoria e a prática, encontrando aplicações em inúmeros problemas reais. Uma de suas principais vantagens é a simplificação que permitem na formulação de problemas complexos [28], o que eleva o nível de abstração que o projetista tem a considerar. De fato, as técnicas evolutivas promovem a denominada "inteligência emergente" (*emergent intelligence*) [51], pois reduzem ou removem a dependência do conhecimento explícito do problema, favorecendo a emergência de restrições específicas do mesmo.

As técnicas evolutivas têm sido aplicadas em áreas nas quais técnicas especializadas não estão disponíveis. Nessas áreas, o tamanho do espaço de soluções é tal que algoritmos tradicionais de busca são ineficazes. No entanto, quando existirem técnicas especializadas ou determinísticas para resolver um problema particular, estas, provavelmente, terão resultados mais precisos e em menor custo computacional. Aqui, ratifica-se o *no-free lunch theorem* [52], que atesta que um desempenho superior obtido por um algoritmo aplicado a uma classe de problemas é "descontado" por um desempenho inferi-

or quando o mesmo é aplicado a outras classes de problemas.

Nos capítulos 3 e 4 são apresentadas metodologias de projeto de sistemas digitais combinacionais e seqüenciais com o emprego de algoritmos evolutivos, assim como estudos de caso para testar seus desempenhos. O capítulo 5 investiga o projeto de protocolos de comunicação numa abordagem evolutiva apresentando técnicas para gerar protocolos a partir de diferentes especificações iniciais de projeto.

Capítulo 3

Síntese de Circuitos Digitais Combinacionais

3.1 Introdução

Devido a crescente demanda em tamanho e complexidade dos circuitos digitais que estão sendo atualmente projetados, sofisticadas ferramentas de CAD (*Computer-Aided Design*) têm sido disponibilizadas no processo de desenvolvimento desses sistemas. Nos últimos anos, a maior inovação nesse ambiente de projeto foi a introdução de linguagens de descrição de *hardware*, um conceito que, embora antigo, obteve difusão na indústria somente a partir da década de 90 [53].

As linguagens de descrição de *hardware* permitem, na sua maioria, a descrição de um projeto eletrônico em vários níveis de abstração, principalmente nos domínios estrutural e comportamental. A descrição do projeto no domínio comportamental é o primeiro passo para o processo de *síntese comportamental* que produzirá na saída o *hardware* desejado. Embora representem um avanço irreversível para gerenciar a complexidade dos atuais projetos, as técnicas atuais de síntese comportamental são, em sua maioria, ineficientes do ponto de vista da qualidade da implementação.

Este capítulo apresenta uma metodologia de síntese de circuitos combinacionais baseada em ferramentas de *síntese de alto nível* [14] e em técnicas de *hardware evolutivo* (EHW: *Evolvable Hardware*) [54, 55], que propõe uma solução para a otimização do uso de recursos em projetos iniciados com descrições em alto nível de abstração. Na seção 3.2 são apresentados os trabalhos relacionados. A seção 3.3 define a síntese comportamental de circuitos digitais e discute a qualidade do *hardware* gerado utilizando esta técnica. A metodologia, apresentada na seção 3.4, utiliza programação genética como paradigma de aprendizado para explorar automaticamente o espaço de

como paradigma de aprendizado para explorar automaticamente o espaço de soluções do projeto e otimizar o uso de recursos. Uma experiência de projeto de um *datapath*, apresentada na seção 3.5, testa o desempenho da metodologia.

3.2 Trabalhos Relacionados

Os algoritmos evolutivos têm sido empregados nos níveis mais baixos do projeto de circuitos eletrônicos, incluindo posicionamento (*placement*) e interligação de células (*routing*) [56, 57], bem como na automação do projeto estrutural desses circuitos [58]. Mais recentemente, os algoritmos evolutivos têm sido utilizados na busca por implementações com requisitos de qualidade. Essa seção descreve os principais trabalhos neste contexto.

Trabalhos recentes [29, 59, 60, 61] apresentam estruturas evoluídas inteiramente em *software* que utilizam simulação para avaliar soluções intermediárias. Essa abordagem é conhecida como *evolução extrínseca* ou *evolução indireta*, em contraste com o trabalho de Thompson [62]. Em [59] é apresentado um algoritmo que evolui circuitos aritméticos a partir de uma matriz de células lógicas de FPGAs (*Field Programmable Gate Arrays*), sendo capaz de redescobrir implementações convencionais para o somador de 2 bits e de, eventualmente, reduzir o número de portas lógicas utilizadas na construção do multiplicador de 2 bits, se comparado às técnicas com intervenção humana. Nessa abordagem, circuitos funcionais com área otimizada foram encontrados por acaso, isto é, sem recorrer a funções de avaliação com múltiplos objetivos.

Em [60] é proposta uma função de aptidão com múltiplos objetivos onde restrições de área têm de ser atendidas ao lado da funcionalidade do circuito. Essa função possui dois estágios. Inicialmente são buscadas soluções 100% corretas, o que ocorre quando a tabela verdade é comprovada. Em seguida, o número de portas lógicas utilizadas no circuito é incorporado à função de aptidão. Essa abordagem utilizou um AG para evoluir um somador binário de 1 bit incluindo a sinalização "vai um" e um multiplicador de 2 bits. Em [29], uma função com múltiplos objetivos, similar à apresentada em [60], é abordada. A principal diferença com a anterior é o uso de uma técnica baseada em popu-

lação para dividir a busca entre várias sub-populações, cada uma delas admitindo um único objetivo, por exemplo, uma saída desejada. Quando estes objetivos são alcançados, suas correspondentes sub-populações são combinadas, o que contribui para minimizar a divergência produzida entre os circuitos codificados e a tabela verdade.

Em [61] é apresentado um ambiente de projeto no qual restrições de desempenho têm de ser atendidas ao lado da funcionalidade do circuito. Nesse ambiente, não apenas portas lógicas, mas blocos funcionais, como por exemplo, o meio-somador, são disponibilizados no processo evolutivo. A principal contribuição deste trabalho é a idéia da utilização de um *único* programa VHDL para evoluir uma população de descrições de *hardware*, o que reduz substancialmente o tempo de compilação. Nesse caso, cada indivíduo é descrito em um processo VHDL distinto. Utilizando essa abordagem, um multiplicador de 3 bits foi evoluído com resultados comparáveis aos obtidos por circuitos desenvolvidos através de ferramentas de CAD.

Seguindo uma outra linha de pesquisa, Thompson [62] desenvolveu um trabalho em que o circuito eletrônico é evoluído e testado em *hardware*, permitindo ao processo evolutivo explorar as propriedades físicas da tecnologia. Tal abordagem ficou conhecida como *evolução intrínseca* ou *evolução direta*. Thompson produziu configurações de FPGA altamente compactas e aparentemente sem coerência, demonstrando a aplicabilidade de sua técnica a pequenos circuitos. No entanto, seus experimentos mostraram que tal técnica não é suficientemente robusta para ter interesse prático. As configurações geradas não funcionaram quando carregadas em componentes distintos, ou quando os componentes estiveram em ambientes com temperaturas diferentes das utilizadas na evolução. Montana *et al*. [63] apontam duas desvantagens de abordagens que utilizam evolução intrínseca: não são portáveis e não escalam para problemas de interesse prático.

Uma alternativa às abordagens acima mencionadas é a utilização de gramáticas formais para representar a evolução de *hardware*, como no trabalho de Hemmi *et al*. [64]. Nesta abordagem, especificações de *hardware* são geradas automaticamente como programas em uma linguagem de descrição de hardware. Os autores utilizaram PG para evoluir árvores a partir de produções para criar descrições de *hardware*. A aptidão é avaliada através de ferramentas de simulação e a implementação é gerada somente *após* o

processo de aprendizado ser concluído. Este sistema foi utilizado para gerar automaticamente um somador binário seqüencial.

3.3 Síntese Comportamental: o Problema da Otimização

Atualmente, o projeto de circuitos integrados (CIs) tem objetivado atender às necessidades do mercado por projetos com elevado nível de complexidade, no menor espaço de tempo possível. Para alcançar tal objetivo faz-se necessário automatizar todo o projeto, desde sua concepção até o produto final. Essa filosofia de projeto, conhecida como *descreva-e-sintetize* (*describe-and-synthesize*) [14], utiliza uma abordagem *top-down* na qual a funcionalidade do sistema é descrita através de um modelo comportamental, simulada para verificar sua correta operação e implementada utilizando ferramentas de síntese de alto nível (HLS: *High-Level Synthesis*).

A síntese comportamental de circuitos digitais (*behavioral synthesis*), ou síntese de alto nível, pode ser definida como o processo de produzir um modelo estrutural no nível de transferência de registradores (RTL: *Register-Transfer Level*) a partir de uma descrição comportamental do sistema [65]. Ela provê uma maneira efetiva de tratar a complexidade demandada pelos circuitos atuais, proporcionando a redução em uma ordem de magnitude do número de objetos que o projetista tem a considerar e, por conseguinte, encurtando o ciclo de projeto [66]. O tempo a mais disponível pode ser utilizado para investigar implementações alternativas através de ferramentas de síntese automáticas. Este procedimento é conhecido como *exploração do espaço de soluções* [67].

O processo da criação de um CI compreende várias etapas (figura 3.1), a primeira delas consistindo na especificação da funcionalidade ou algoritmo do sistema. Há uma variedade de modelos conceituais que podem ser utilizados para este propósito, como detalhado em [66]. Máquinas de estados finitos (MEFs), MEFs com fluxo de dados (FSMDs: *Finite-State Machines Datapath*) e Redes de Petri são alguns exemplos. Linguagens de programação são modelos conceituais de interesse particular, visto que são capazes de expressar simultaneamente dados, atividade e controle [15]. Após a especificação da funcionalidade do sistema, é realizada a síntese comportamental, que trans-

forma o modelo funcional em um projeto RTL. Em seguida, são realizadas as sínteses lógica e de *layout*.

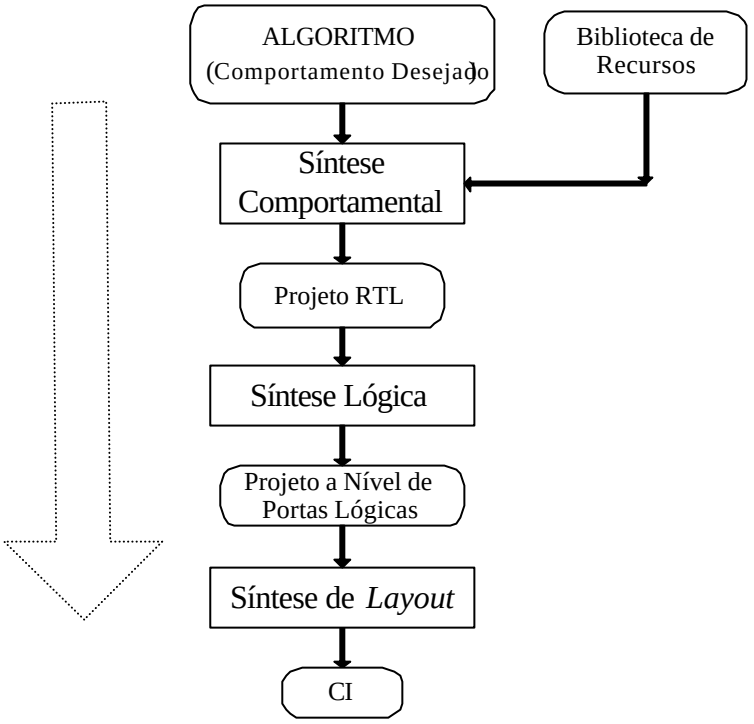


Figura 3.1: Processo de criação de um CI a partir de uma descrição comportamental.

Linguagens de programação para a descrição de *hardware*, mais conhecidas como linguagens de descrição de *hardware* (HDLs: *Hardware Description Languages*), tal como VHDL (*Very High-Speed Integrated Circuit* (VHSIC) HDL) [53, 68], capturam a funcionalidade do sistema de forma adequada para manipulação por computadores. VHDL é uma notação formal desenvolvida para ser utilizada em todas as fases da criação de projetos eletrônicos. Este formalismo suporta documentação, verificação, descrição em vários níveis de abstração, síntese e teste de projetos eletrônicos, sendo amplamente aceito em sistemas EDA (*Electronic Design Automation*). Em contrapeso, VHDL possui semântica projetada com intuito de simulação, implicando que apenas um subconjunto de suas instruções possa ser sintetizado.

Embora, atualmente, existam muitas ferramentas de CAD disponíveis, possibilitando o projeto de circuitos eletrônicos em alto nível de abstração, o mercado ainda resiste

em adotá-las. Gajski *et al*. [65] apontam duas razões para este fato: baixa qualidade de resultados e ausência de interação entre o projetista e a ferramenta *durante* o processo de síntese. Especificações em alto nível de abstração são normalmente consideradas "pouco sintetizáveis", vez que, nesse nível de abstração, muitos detalhes de implementação são mascarados do projetista [69]. Para resolver este problema, as ferramentas de síntese restringem o espaço de soluções assumindo uma arquitetura alvo na qual todas as especificações em alto nível são mapeadas. O resultado é uma solução localmente otimizada para cada projeto.

Por outro lado, implementações em *hardware* têm sido consideravelmente simplificadas pela prototipagem com FPGAs. Estatísticas recentes indicam que, atualmente, mais da metade dos projetos de CIs é iniciada utilizando FPGAs [70]. As FPGAs eliminam a necessidade de percorrer as etapas convencionais de produção de um CI, reduzindo o risco financeiro e o tempo de desenvolvimento. Adicionalmente, modernas ferramentas de síntese oferecem suave migração entre as tecnologias FPGA e ASIC (*Application Specific Integrated Circuit*), proporcionando mapeamento direto entre projetos com FPGAs e bibliotecas ASIC [71].

3.4 Metodologia

A metodologia de síntese de circuitos digitais combinacionais [7, 8, 9] evolui uma população de programas VHDL buscando uma solução (descrição de *hardware*) que, conjuntamente, implemente a funcionalidade desejada e satisfaça a restrição de área. Esta metodologia é mostrada na figura 3.2 onde se notam dois de seus principais componentes: o núcleo do sistema de PG, representado pelo bloco GPK, e a função de avaliação de aptidão, envolvendo várias rotinas com funções específicas. Diferentemente de outras abordagens [61, 64], a compilação e o *place-and-route* (posicionamento e interligação de células) são realizados para cada individuo da população, fornecendo ao sistema a informação exata do uso de recursos.

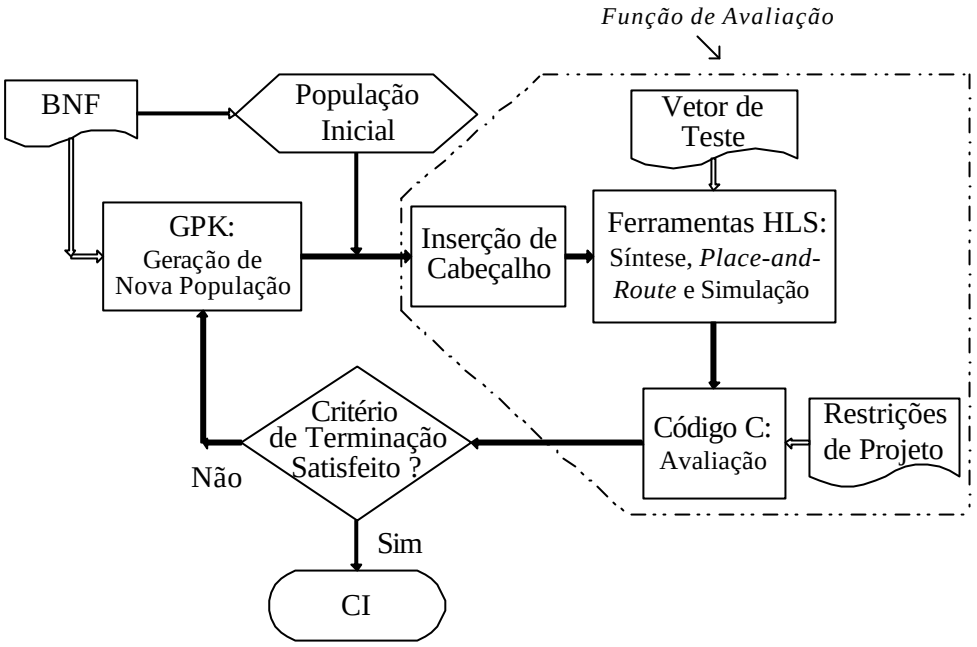


Figura 3.2: Metodologia de síntese de circuitos digitais combinacionais.

Inicialmente, uma população de programas é gerada de modo aleatório com a restrição dada por uma descrição BNF pré-definida, contendo um subconjunto da gramática VHDL. Na seqüência, uma função objetivo é utilizada para avaliar todos os indivíduos (genomas) e atribuir um valor de aptidão a cada um deles. O sistema prossegue com a aplicação de operadores genéticos (bloco GPK), gerando novas populações até que o critério de terminação seja satisfeito. Uma vez que a presente metodologia diz respeito a soluções desconhecidas, o sistema termina quando o número de gerações se iguala a um número máximo pré-determinado (predicado de geração).

Para criar o código fonte VHDL, a declaração da entidade e a parte de declaração da arquitetura de um programa VHDL são mantidas fixas, pois são comuns a todos os indivíduos da população. Somente a parte de procedimentos do código VHDL é afetada pelo processo evolutivo. O método de geração de programas VHDL é um outro ponto em que a metodologia difere da abordagem proposta em [61]. Na presente metodologia, descrições de circuitos em VHDL são geradas através de programação genética. Em [61] foi utilizado um algoritmo genético com genomas de tamanho fixo para evoluir estruturas. A partir de então, estas estruturas foram instanciadas em um programa VHDL através da cláusula *port map*, gerando uma descrição VHDL do circuito no nível estru-

tural (esquemático textual).

O cálculo da função objetivo, ou função de avaliação de aptidão, é um procedimento chave em um sistema de PG. As etapas envolvidas nesta operação são mostradas na figura 3.3. Após a geração de uma população é necessário completar o código fonte VHDL de cada indivíduo inserindo a declaração da entidade e a parte de declaração da arquitetura. Em seguida, cada indivíduo é sintetizado utilizando ferramentas de síntese de alto nível. Para calcular efetivamente a aptidão de cada indivíduo faz-se necessário simular cada um dos casos de aptidão, isto é, casos de treinamento, e checar estes resultados com os valores desejados.

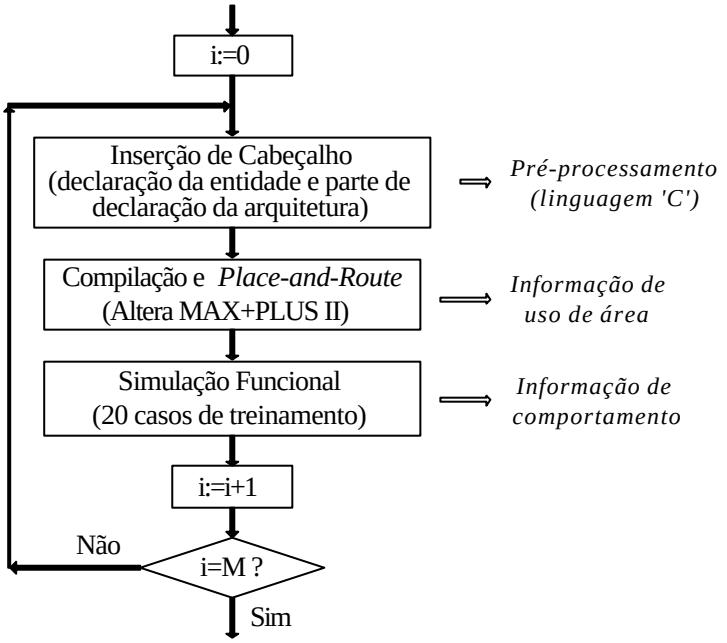


Figura 3.3: Bloco função de avaliação da figura 3.2.

Função Objetivo

A função a ser minimizada é uma função de erro com múltiplos parâmetros, combinando funcionalidade e área em uma abordagem com dois estágios similar à adotada em [60]. Assim como na abordagem proposta por Kalganova *et al*. (vide seção 3.2), na presente metodologia a função objetivo assume duas definições, cada uma delas utilizadas em um estágio distinto da evolução: inicialmente, a especificação funcional (F_1) deve

ser atendida; em seguida, o sistema busca por implementações, funcionalmente corretas, com uso otimizado de recursos (F_2).

Seja o erro médio de funcionalidade, fe_{av}, definido por:

$$fe_{av} = \frac{1}{N}\sum_{i=1}^{N} w_i \left| D_i - S_i \right| \tag{3.1}$$

onde N é o número de casos de treinamento, w_i é um fator de ponderação, D_i é o valor desejado para o caso de treinamento i e S_i é o valor obtido (simulado) para o caso de treinamento i. A função objetivo ou função de aptidão F, utilizada na metodologia, tem a forma:

$$F = \begin{cases} F_1 = fe_{av} & if\ \exists i \in \{1,\cdots,N\} \mid \left| \dfrac{D_i - S_i}{D_i} \right| > te \\ F_2 = fe_{av} * k(lu) \end{cases} \tag{3.2}$$

onde te é o erro de especificação funcional desejado para os casos de treinamento, lu é o número de elementos lógicos utilizados e k é uma função de lu. A função $k(lu)$ é um fator de ponderação que considera o efeito da área utilizada para implementar a funcionalidade desejada. $k(lu)$ é uma função exponencial de lu (*logic units*) que privilegia indivíduos contendo as menores quantidades de elementos lógicos, assumindo a forma:

$$k(lu) = ae^{b*lu} \tag{3.3}$$

onde a e b são constantes que devem ser definidas resolvendo o seguinte sistema:

$$\begin{aligned} k_{\min} &= ae^{b*lu_{\min}} \\ k_{\max} &= ae^{b*lu_{\max}} \end{aligned} \tag{3.4}$$

onde k_{min} e k_{max} são os valores mínimo e máximo de $k(lu)$ e lu_{min} e lu_{max} são a mínima e a

máxima quantidade de elementos lógicos, respectivamente.

A função de aptidão F, definida pela equação 3.2, é igual ao erro médio de funcionalidade, isto é, $F = F_1 = fe_{av}$, caso pelo menos um dos casos de treinamento apresente erro relativo superior ao erro de especificação funcional te. A comutação de F_1 para F_2 ocorre quando o indivíduo que está sendo avaliado apresenta erro relativo igual ou inferior a te para todos os casos de treinamento. Então, o fator de ponderação $k(lu)$, que assume valores entre 0,5 e 1,0, multiplica fe_{av}.

Descrição BNF

A descrição BNF a ser fornecida ao sistema deve contemplar a propriedade de suficiência. Além disso, a BNF deve possibilitar às descrições VHDL uma flexibilização em relação ao número de elementos estruturais que aproximarão a função desejada. Esta flexibilização é mostrada através do exemplo:

```
<code>   ::=   <line> | <code> <line>;
<line>   ::=   <expr>;
<expr>   ::=   <expr> <op> <expr> | <pre-op> "(" <expr> ")" | <var>;
```

onde o símbolo não terminal <*op*> representa operações matemáticas e os símbolos não terminais <*var*> e <*pre-op*> representam uma variável e uma função com um argumento, respectivamente. Utilizando as produções na descrição BNF acima definida é possível construir um programa constituído por uma ou mais linhas de código, cada uma delas descrevendo uma ou mais estruturas funcionais.

Os parâmetros de PG devem ser ajustados para o rendimento máximo do algoritmo. A tabela 3.1, apresentada na subseção 3.5.3, atribui valores para estes parâmetros em um estudo de caso específico.

3.5 Estudo de Caso: Síntese Otimizada de *Datapath* em FPGA

Sob o ponto de vista de arquitetura, o projeto de um circuito digital pode ser dividido em duas partes principais [14]: a unidade de controle, na maioria das vezes modelada por uma máquina de estados finitos; e o fluxo de dados (*datapath*), que implementa os blocos funcionais que afetam os dados de entrada. O presente projeto consiste em sintetizar automaticamente um *datapath* com o emprego de programação genética para gerar procedimentos VHDL. A função a ser implementada pelo *datapath* é avaliada através da função de aptidão definida pela equação 3.2, que leva em consideração o número de células lógicas utilizadas para implementar o circuito.

Para a obtenção dos resultados numéricos mostrados a seguir foi utilizado o sistema de programação genética GPK versão 1.22 [36] e foram desenvolvidos programas na linguagem de programação 'C' que implementaram os procedimentos de pré-processamento e as chamadas de rotina (compilação e simulação) descritas na figura 3.3. Foram utilizadas ferramentas de síntese de alto nível MAX+PLUSII [72]. O sistema, que é totalmente automatizado, é executado em um microcomputador com arquitetura X86 de 1GHz de relógio e 256MB de memória RAM, sob o sistema operacional Windows98.

3.5.1 Descrição do Projeto

O experimento consiste em sintetizar automaticamente um *datapath* que realiza uma aproximação para a Distância Euclidiana entre dois pontos em um espaço bidimensional, literalmente a raiz quadrada da soma do quadrado de dois números. A Distância Euclidiana é freqüentemente utilizada para medir a distância, ou similaridade, entre dois casos em abordagens de reconhecimento de padrões, bem como em algoritmos de computação gráfica. A função que se deseja implementar foi obtida em [65], que utilizou a fórmula:

$$\sqrt{a^2 + b^2} \cong max((0{,}875x+0{,}5y),x) \quad (3.5)$$

onde $x = max(|a|,|b|)$ e $y = min(|a|,|b|)$.

Nesse problema de *regressão simbólica múltipla* [5] o objetivo é encontrar uma função com duas variáveis independentes, *a* e *b*, na forma simbólica, que possibilite a aproximação desejada para um grupo de 20 conjuntos de pontos numéricos. A figura 3.4 mostra a "caixa preta" do projeto do *datapath*. Nessa figura, os sinais de entrada *start* e de saída *done* estão presentes para exercer funções de controle.

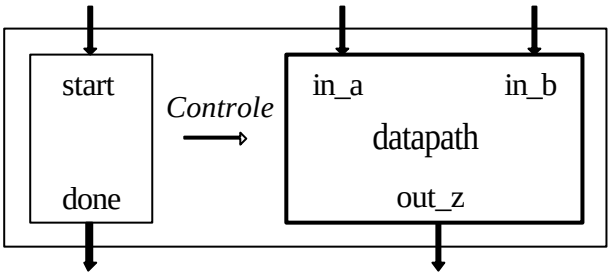


Figura 3.4: "Caixa preta" do projeto do *datapath*.

3.5.2 Definição da BNF

A seguinte BNF foi definida para resolver o problema em questão:

```
S     ::= <fe>;
<fe>  ::= "PROCESS BEGIN WAIT UNTIL (clk'event and clk='1'); IF
          in_a>=in_b THEN x<=in_a; y<=in_b; ELSE y<=in_a; x<=in_b;
          END IF; t6<=" <expr> ";IF t6>=x THEN t7<=t6; ELSE t7<=x;
          END IF; out_z<=t7; done<='1'; END PROCESS;"
<expr>::= "("<expr>")"<op>"("<expr>")" | <var> |
                <var> "(7 downto" <shf_size>")";
<op>  ::= " + " | " – ";
<var> ::= " x " | " y " | " w ";
<shf_size> ::= "1" | "2" | "3" | "4" | "5" | "6" | "7";
```

Nesta descrição BNF, *x*, *y*, *w*, *t6* e *t7* são sinais internos à arquitetura do programa

VHDL. O sinal *w*, ao qual se atribui zero, está presente para facilitar a inserção e remoção de um símbolo terminal. As cláusulas *IF* implementam as operações *max* e *min* da equação 3.5. O problema original foi restringido a utilizar operandos de entrada e saída como *unsigned* (8 bits) ao invés de inteiro (32 bits), reduzindo o custo computacional.

3.5.3 Principais Atributos de PG

As principais características para o primeiro estágio (F_1) do sistema de PG estão definidas na tabela 3.1.

Tabela 3.1: Características de PG para o primeiro estágio (F_1) do projeto do *datapath*.

Objetivo:	Criar um programa VHDL que implemente uma função de dois argumentos e que satisfaça uma amostra de 20 conjuntos $\{a_i, b_i, F(a_i,b_i)\}$, onde a função alvo seja definida por $F(a,b) = sqrt(a^2 + b^2)$.
Símbolos terminais:	*x*, *y*, *w*, +, –, *7 downto* (*operador lógico deslocar à direita*), *1*, *2*, *3*, *4*, *5*, *6*, *7* (*índice do operador lógico deslocar à direita*) e a sentença entre aspas no lado direito de *<fe>*.
Símbolos não terminais:	*<fe>*, *<expr>*, *<op>*, *<var>*, *<shf_size>*.
Casos de treinamento:	Amostra de 20 conjuntos de dados $\{a_i, b_i, F(a_i,b_i)\}$, a_i e b_i escolhidos aleatoriamente a partir do intervalo [0, 255], satisfazendo a restrição $sqrt(a_i^2 + b_i^2) \leq 255$.
Aptidão:	A média, a partir de 20 casos de treinamento, do valor absoluto da diferença entre o valor da variável dependente produzida pela simulação de programas VHDL sintetizados e o valor alvo da variável dependente (Cf. equações 3.1, 3.2, 3.3 e 3.4).
Método de seleção:	Proporcional à aptidão.
Parâmetros principais:	M=100, G=51, p_c=0,73 (cruzamento em um ponto), p_r=0,15, p_m=0,12, sem elitismo.

No segundo estágio da evolução (F_2) os parâmetros de PG permaneceram inalterados, exceto as linhas “objetivo” e “aptidão” da tabela 3.1 que modificaram para incorporar a busca por implementações com recursos otimizados.

3.5.4 Resultados

Para a realização deste experimento foi utilizado o componente FPGA EPF8282A da Altera [72], contendo 282 elementos lógicos. Cada elemento lógico consiste em uma tabela de *look-up*, isto é, um gerador de funções que computa qualquer função de quatro variáveis, e um registrador programável. Neste experimento, a área será medida em números de elementos lógicos. O erro funcional (*te*) foi definido em 2,5% (Cf. equações 3.1 e 3.2). Ele estabelece o erro máximo entre a variável dependente, resultante da simulação de programas VHDL evoluídos, e o valor desejado da variável dependente para cada um dos casos de treinamento.

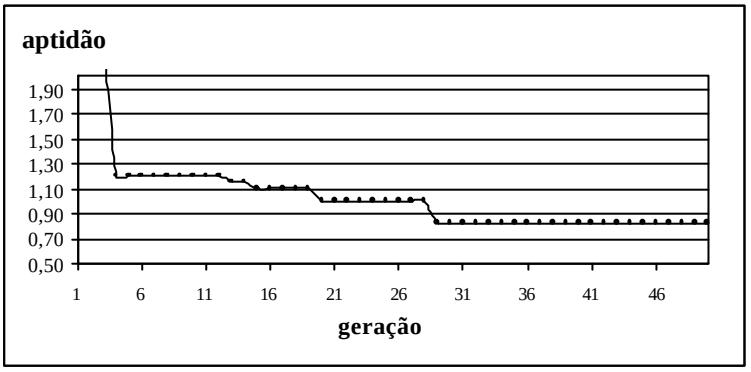
Foi utilizada uma população de 100 indivíduos que evoluiu até um máximo de 51 gerações. Este experimento utilizou $N = 20$, $w_i = 1$ e $k(lu) = 0{,}25e^{(0{,}0139*lu)}$, que garante $k(lu) \in [0.5,1.0]$ para $lu \in [50,100]$ (Cf. equações 3.3 e 3.4). A população inicial foi gerada através do método *ramped half-and-half* [5], substituindo o procedimento de inicialização original do GPK. O método *ramped half-and-half* garante a diversidade da população de indivíduos através de uma ponderação feita entre outros dois métodos básicos de inicialização de árvores: os métodos *full* e *grow*. O método *full* cria indivíduos com o objetivo de atingir a profundidade máxima da árvore, sendo esta definida como o número de nós pertencente ao maior caminho direto (*non-backtracking*) entre o nó raiz e uma folha [36]. O método *grow* gera indivíduos de formas e tamanhos variados através de uma seleção aleatória de valores para a profundidade máxima da árvore.

O experimento foi realizado com 10 execuções independentes. A figura 3.5a mostra a curva de aptidão do melhor indivíduo em uma execução particular. Na geração 18 dessa execução surgiu o primeiro indivíduo funcional, ou seja, um programa VHDL no qual todos os casos de treinamento atenderam a especificação funcional do erro ($te = 2{,}5\%$). A partir deste ponto, o sistema iniciou a busca por implementações otimizadas. Durante o segundo estágio da evolução vários indivíduos atenderam à especificação funcional, embora com uso não otimizado de recursos. Indivíduos funcionais com 90, 92 e 95 elementos lógicos foram encontrados. Somente na geração 29 o melhor indivíduo emergiu. Sua aptidão (F) foi de 0,827 e o erro médio de funcionalidade (fe_{av}), 1,103. Este indivíduo foi sintetizado com 78 elementos lógicos. Um trecho da parte de

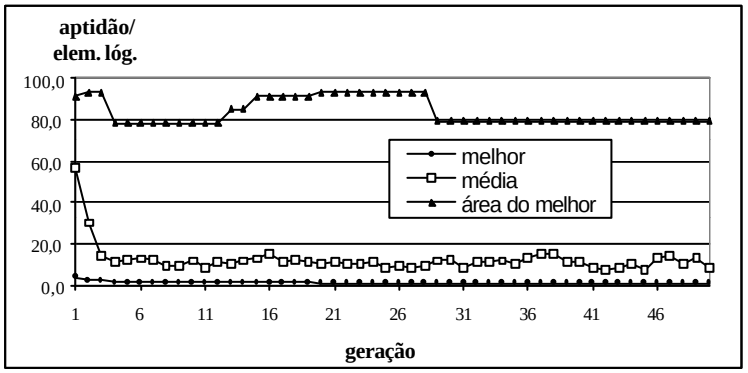
arquitetura do programa VHDL deste indivíduo é mostrado abaixo:

```
BEGIN
  w <="00000000";
  PROCESS BEGIN WAIT UNTIL (clk'event and clk='1');
  IF in_a >= in_b THEN x <= in_a; y <= in_b;
  ELSE y <= in_a; x <= in_b; END IF;
  t6 <= ((y(7 downto 1)) + ((x) - (x(7 downto 3)))) + (y(7 downto 6));
  IF t6 >= x THEN t7 <= t6;
  ELSE t7 <= x; END IF;
  out_z <= t7; done <= '1';
END PROCESS;
```

A figura 3.5b mostra, em destaque, a evolução da área do melhor indivíduo. Observa-se, nesta figura, que o melhor indivíduo aumenta sua área a partir da geração 13 com o objetivo de melhorar a aproximação da função. Na geração 18 o melhor indivíduo atinge a aproximação desejada e inicia-se a busca por implementações com área otimizada, o que é alcançado na geração 29.



a)



b)

Figura 3.5: (a) Histograma da aptidão; e (b) da área do melhor indivíduo.

A tabela 3.2 mostra um quadro comparativo entre a área utilizada e a precisão da aproximação da função para outros valores de *te*. Esta tabela revela que o indivíduo com área ótima para *te* = 25% não utilizou o sinal interno *y*, o menor entre as duas entradas. Isto indica que o sistema busca, de fato, por implementações com área otimizada, levando em conta a quantidade de recursos e o roteamento utilizados pela ferramenta de síntese.

Tabela 3.2: Quantidade de elementos lógicos para outros valores de *te* com respectivas descrições de *hardware*.

te	Elementos lógicos	Descrição de *hardware*
25%	51	`x + x(7 downto 3)`
15%	58	`x + y(7 downto 2)`
12%	59	`x + y(7 downto 1)`
5%	70	`x - x(7 downto 3) + y(7 downto 1)`

Em algumas execuções notou-se que o primeiro indivíduo funcional apresentava também a menor área. No entanto, isto nem sempre se comprovou, como mostrado para o caso de *te* = 2,5%. Alternativamente, realizou-se uma experiência adicionando o operador matemático *multiplicação* e o operador lógico *deslocar à esquerda* no lado direito das produções que definem as substituições para os símbolos não terminais *<op>* e *<expr>*, respectivamente. O sistema mostrou-se robusto, eliminando, nas primeiras gerações, indivíduos contendo estes dois operadores.

Como desvantagem, a metodologia apresentada mostrou-se lenta. As ferramentas atualmente disponíveis para a compilação de programas VHDL exigem alto custo computacional, particularmente para funções *place-and-route* [63]. Uma execução para uma população de 100 indivíduos consumiu tempo médio de 3 min por geração num microcomputador com arquitetura X86 de 1GHz de relógio e 256MB de memória RAM, sob o sistema operacional Windows98. Nesse caso, para obter o melhor indivíduo foram necessários 29 × 3 min = 1 h e 27 min de processamento em uma execução particular (Cf. seção 3.5).

Para contornar este problema foi introduzido um pré-estágio a ser executado antes

do início do processo principal de otimização (figura 3.2). Nesse pré-estágio o sistema evolui um único programa VHDL, como em [61], com objetivo único de induzir toda a população a aproximar-se da funcionalidade desejada. Conseqüentemente, a função *place-and-route* é eliminada neste pré-estágio, que termina quando pelo menos dois indivíduos funcionais surgem. Essa alteração na metodologia aumentou significativamente o desempenho do sistema.

3.6 Comentários

Neste capítulo foi apresentada uma metodologia de síntese de circuitos digitais combinacionais através de programação genética e de ferramentas de síntese de alto nível. Esta metodologia evolui uma população de programas VHDL para gerar soluções que atendam à funcionalidade especificada, ao mesmo tempo em que busca por implementações com uso otimizado de recursos. A abordagem apresentada é fundamentada em uma compilação de técnicas, descritas na seção 3.2, utilizadas em diferentes abordagens de *hardware* evolutivo encontradas na literatura.

A grande vantagem da metodologia proposta neste capítulo é sua habilidade em gerar *hardware* otimizado a partir de uma descrição feita no domínio comportamental. Essa descrição utiliza meramente uma expressão matemática que relaciona saídas a entradas. Por outro lado, é importante notar que o trecho do programa passível de evolução é uma descrição estrutural do projeto, afetando, portanto, o número de elementos lógicos utilizados. A propriedade de suficiência deve ser atendida na definição da BNF, ou seja, seus símbolos terminais (operações e operandos) devem ser convenientemente escolhidos pelo projetista com base no problema que se deseja resolver. Esta propriedade também é utilizada por outros paradigmas de aprendizado automático [5].

A desvantagem detectada no uso desta abordagem foi o excessivo tempo despendido na compilação de programas VHDL, em especial na etapa *place-and-route*. A solução proposta para este problema, ou seja, a utilização de um único programa VHDL contendo todos os indivíduos até que dois indivíduos funcionalmente corretos surjam, em contraste com a utilização de um programa VHDL para cada indivíduo (primeira versão),

reduziu o tempo total de processamento em cerca de 30%.

No capítulo a seguir são apresentadas abordagens de síntese de sistemas seqüenciais a partir de seqüenciais parciais de entrada e saída.

Capítulo 4

Síntese de Sistemas Seqüenciais a Partir de Seqüências Parciais de Entrada e Saída

4.1 Introdução

O projeto de sistemas digitais compreende as áreas de sistemas combinacionais e de sistemas seqüenciais. Embora a maioria das aplicações de técnicas evolutivas em projetos de circuitos tem compreendido a área de circuitos combinacionais, a aplicação de tais técnicas no projeto de sistemas seqüenciais, que contêm redes de realimentação, pode ser considerada ainda mais promissora devido à dificuldade que as técnicas convencionais encontram em tratar projetos deste tipo [73].

Os métodos convencionais de síntese de sistemas seqüenciais requerem o conhecimento prévio do comportamento do circuito na forma de um diagrama de estados [74]. Tais métodos não podem ser utilizados quando toda a informação do sistema se resume a seqüências parciais de entrada e saída. Em outras palavras, deseja-se modelar um sistema já existente, na forma de uma "caixa preta", e tem-se acesso somente às suas entradas e saídas. Nesses casos, as técnicas convencionais empregadas pelas ferramentas de síntese, tal como SIS [75], não podem ser diretamente utilizadas [73, 74].

A tarefa de modelar sistemas já existentes exclusivamente pela observação de seus comportamentos de entrada e saída é de grande interesse quando o propósito é: (a) inferir os estados de sistemas naturais; (b) modelar sistemas artificiais que foram construídos a partir de descrições informais ou para os quais perderam-se as documentações; e (c) gerar novos sistemas a partir de especificações descritas em alto nível de abstração, por exemplo, através de *casos de uso* (*use cases*). Sistemas meteorológicos são exemplos da primeira aplicação. As aplicações (b) e (c) referem-se às abordagens de enge-

nharia reversa (*reverse engineering*) e de engenharia direta (*forward engineering*), respectivamente.

A engenharia reversa analisa um sistema existente para identificar seus componentes e seus relacionamentos e criar representações de sistemas em uma outra forma ou em um nível de abstração maior. Por outro lado, o modelo de um sistema pode não revelar o comportamento dinâmico do mesmo, como por exemplo, situações de concorrência. Nesses casos, a engenharia reversa, que revela o comportamento do sistema em tempo de execução, pode ser utilizada para complementar a informação fornecida pelo modelo original, essencialmente de natureza estática [76].

A engenharia direta traduz o processo tradicional de transformar abstrações de alto nível ou projetos lógicos, com informações independentes de implementação, em realizações físicas do sistema. A geração de entidades de protocolo de comunicação a partir de descrições intuitivas, como cenários de interação ou casos de uso, que expressam o comportamento do sistema sem revelar sua estrutura interna, é um exemplo de utilização das técnicas tratadas neste capítulo em abordagens de engenharia direta (vide seção 5.4).

Neste capítulo são apresentadas metodologias que utilizam técnicas evolutivas para sintetizar sistemas seqüenciais a partir de seqüências de entrada e saída observáveis. Na seção 4.2 são apresentados os trabalhos relacionados. A seção 4.3 define as máquinas de estados finitos. Na seção 4.4 é apresentada a metodologia básica, que é modificada, na seção 4.5, para gerar máquinas de estados finitos com número mínimo de estados. A seção 4.6 descreve uma abordagem para tratar a convergência prematura nesses sistemas. Por fim, na seção 4.7 são apresentados e analisados os resultados de três experimentos relacionados à síntese de circuitos lógicos seqüenciais síncronos de pequeno porte.

4.2 Trabalhos Relacionados

A idéia de evoluir autômatos a partir de eventos observáveis tem sido divulgada desde os anos 60. Uma das primeiras experiências nesse sentido foi conduzida por Fogel [20] e Fogel *et al*. [21] utilizando uma técnica que pela primeira vez empregou o concei-

to de população. Esta técnica, que ficou conhecida como programação evolutiva, é uma das técnicas precursoras das técnicas evolutivas, não se enquadrando inteiramente nesta última por não utilizar o operador genético de cruzamento. Apesar de restrito a problemas envolvendo um pequeno espaço de busca [77], o experimento de Fogel propiciou o desenvolvimento de autômatos que predizem a próxima saída baseados em seqüências de entrada conhecidas.

Trabalhos desenvolvidos na última década obtiveram sucesso em sintetizar circuitos seqüenciais com a ajuda de AGs, tais como [74, 78, 79]. Em [74], uma abordagem para sintetizar circuitos lógicos seqüenciais a partir de seqüências parciais de entrada e saída é proposta. Máquinas de estados finitos de pequeno porte foram sintetizadas em FPGA, tais como divisores de freqüência e somadores seriais. Os trabalhos [78, 79] utilizaram representação baseada em estado como alternativa à representação baseada em tecnologia, isto é, um diagrama de conexões entre portas lógicas e registradores, empregada em [74].

Na representação baseada em estado, o próximo estado e a saída correspondente a cada par estado atual/entrada são codificados em uma *string* definida como o *cromossomo*. Nesse caso, já que o número de estados do autômato é desconhecido, um grande número de estados é utilizado resultando em MEFs com estados redundantes e/ou não alcançáveis. Em [79] foram calculados e verificados empiricamente o número e o comprimento das seqüências de entrada e saída que garantem a correção semântica, isto é, o comportamento desejado da MEF.

Embora a representação baseada em estado não possua a restrição de portabilidade imposta pela representação baseada em tecnologia, ela não é eficiente para sistemas com grande número de estados uma vez que provê uma ordenação temporal completa de todas as distinções, levando a descrições extensas e complexas [80]. No entanto, certas classes de problemas, como o projeto de protocolos de comunicação, tratado no capítulo 5, podem se beneficiar da ordenação completa dos eventos de entrada e dos estados para garantir uma importante propriedade de segurança.

4.3 Máquinas de Estados Finitos (MEFs)

As metodologias de síntese de sistemas seqüenciais a serem apresentadas neste capítulo utilizam como formalismo de base as MEFs (FSMs: *Finite-State Machines*), que são predominantes em projetos de engenharia e ciências da computação [14]. As MEFs podem ser classificadas de acordo com sua função de saída. Se a função de saída está relacionada somente com o estado atual, a MEF é denominada uma máquina Moore. Caso a função de saída seja dependente, conjuntamente, do estado atual e da (s) entrada (s), a MEF é denominada uma máquina Mealy. É importante notar que a utilização de máquinas Mealy, cujas saídas estão associadas às transições, resulta na implementação de sistemas com igual ou menor número de estados do que com máquinas Moore.

Uma MEF Mealy M é definida formalmente por uma 7-tupla $\{Q, V, T, q_0, V', O, D\}$ onde $Q \neq \emptyset$ é um conjunto finito de estados, V é um conjunto finito de entradas, T é a função de transição de estado, $q_0 \in Q$ é o estado inicial, V' é um conjunto finito de saídas, O é a função de saída e D é o domínio de especificação, sendo este último um subconjunto de $Q \times V$. T e O caracterizam o comportamento da MEF. Se $D = Q \times V$, então T e O são definidos para toda combinação estado atual/entrada possível e, portanto, a MEF é denominada *completamente* (ou *totalmente*) *especificada*. Caso contrário, ela é denominada *parcialmente especificada*.

Uma forma padrão para todas as redes seqüenciais é a implementação canônica, ou implementação de Huffman-Moore [81], que se baseia diretamente na descrição de estados de um sistema. Ela define o próximo estado, $q(t+1)$, e a saída, $v'(t)$, nas formas:

$$q(t+1) = T(q(t), v(t)) \tag{4.1}$$

$$v'(t) = O(q(t), v(t)) \tag{4.2}$$

A implementação canônica da máquina Mealy, cujos componentes são organizados conforme descreve a figura 4.1, consiste em um registrador de estado, que armazena o estado atual, e uma rede combinacional, que implementa as funções de saída e de transição de estado. Em geral, a descrição de um sistema seqüencial inclui um estado inicial no qual é necessário colocar o sistema a fim de obter o comportamento de entrada e saí-

da desejado. Esta inicialização é realizada por uma entrada exclusiva para este propósito.

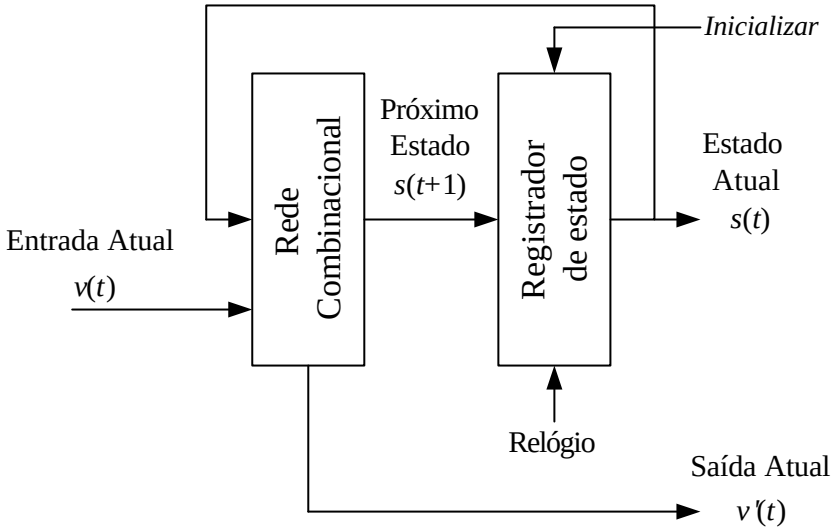


Figura 4.1: Implementação canônica da máquina Mealy.

4.4 Metodologia Básica

A metodologia de síntese de sistemas seqüenciais a partir de seqüências parciais de entrada e saída gera na saída uma MEF que modela um sistema já existente pela observação de seus eventos de interface. A figura 4.2 ilustra a aplicabilidade da metodologia, fornecendo o diagrama de conexões entre o sistema existente (sistema observado) e o autômato a ser evoluído.

As etapas envolvidas nesta metodologia são mostradas na figura 4.3. Inicialmente, uma população de MEFs é criada aleatoriamente a partir de uma descrição BNF especificada de forma a gerar representação baseada em estado, o que corresponde a uma tabela de transição de estados.

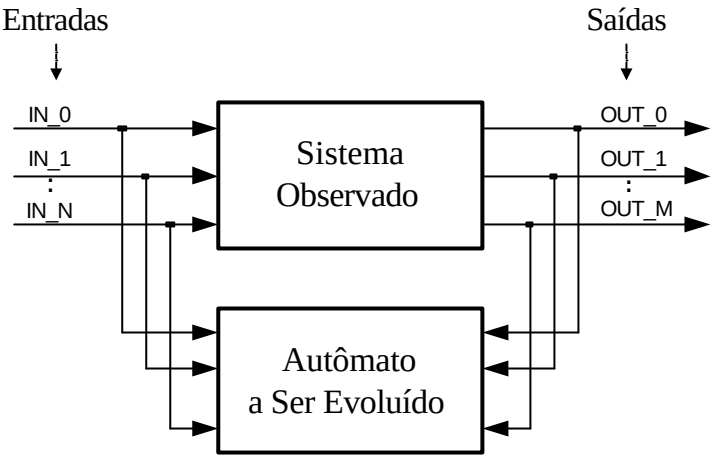


Figura 4.2: Aplicabilidade da metodologia de síntese de sistemas seqüenciais.

Em seguida, cada indivíduo da população é avaliado através de uma função de avaliação que lhe atribui uma aptidão. Caso ao menos um indivíduo apresente a aptidão desejada, indicando que suas saídas são iguais às saídas especificadas quando o autômato é estimulado pelas mesmas seqüências de entrada, o sistema termina exteriorizando a MEF que, *a priori*, modela o sistema (Cf. discussão da correção semântica ao final desta seção). Caso contrário, os operadores genéticos são aplicados (bloco GPK da figura 4.3) e uma nova população é gerada para avaliação.

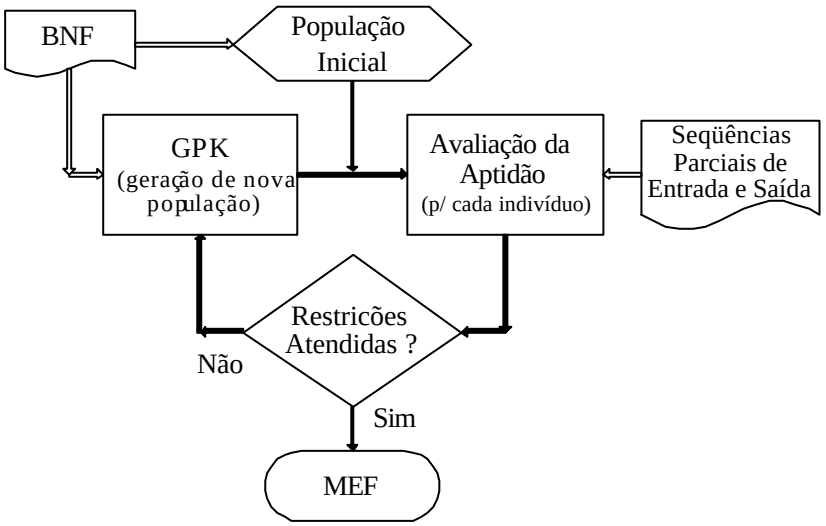


Figura 4.3: Metodologia de síntese de sistemas seqüenciais a partir de seqüências parciais de entrada e saída.

Representação do Fenótipo

O fenótipo utilizado para codificar uma MEF emprega representação baseada em estado. A figura 4.4 mostra esta codificação com S estados e I entradas, totalizando $2 \times I \times S$ genes.

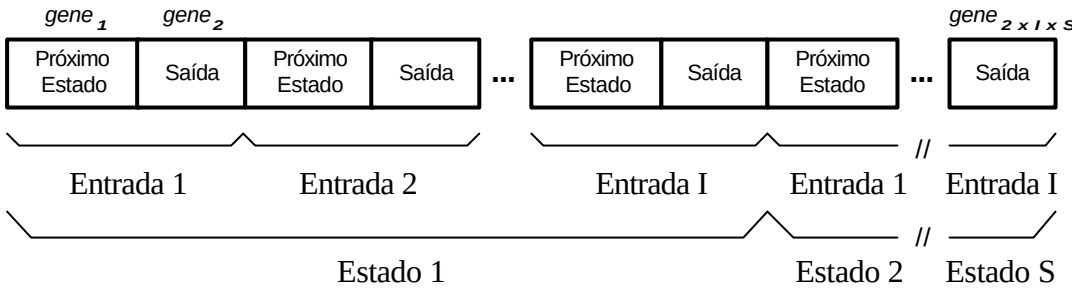


Figura 4.4: Fenótipo com representação baseada em estado.

A forma de representação apresentada na figura 4.4 equivale a representação algorítmica de um autômato através de cláusulas *IF-THEN-ELSE* aninhadas, como se segue:

```
IF estado_1
      THEN
      IF entrada_1 THEN
            proximo_estado:=
            saida:=
      ELSIF entrada_2 THEN ...
      ...
ELSIF estado_2
      THEN
      IF entrada_1 THEN
            proximo_estado:=
            saida:=
      ELSIF entrada_2 THEN ...
      ...
...
END
```

Função de Aptidão

O valor de aptidão atribuído a um dado comportamento da MEF, avaliado através de uma perspectiva de eventos de entrada e saída, é definido pela função de aptidão F na

forma:

$$F = \sum_{i=1}^{N} w_i H_i \tag{4.3}$$

onde N é o número de casos de treinamento ou *seqüências de treinamento*, w_i é o fator de ponderação para a seqüência de treinamento i e H_i é o número de acertos relativos à seqüência de treinamento i. H_i é calculado da seguinte forma: inicialmente a MEF é colocada no estado inicial; em seguida, para cada evento de entrada da seqüência de treinamento i, a saída da MEF é comparada com a saída correspondente da seqüência de treinamento i e um acerto é assinalado em caso de igualdade.

Comprimento das Seqüências de Treinamento

As seqüências de entrada e saída, ou seqüências de treinamento, devem ser longas o suficiente para exercitar todos os caminhos da MEF a ser modelada. Uma seqüência muito curta pode causar ambigüidade na descrição do comportamento desejado. Uma seqüência muito longa, no entanto, pode comprometer o desempenho do sistema.

A solução para o problema de encontrar o comprimento ideal das seqüências de treinamento foi obtida da fórmula derivada em [74], sendo esta baseada na solução para o problema "tempos de espera em amostragem" (*waiting times in sampling*), descrito em [82]. Esta fórmula define o comprimento da seqüência de treinamento, L, como:

$$L = E(S) \times E(I) \tag{4.4}$$

onde $E(S)$ e $E(I)$ são o *valor esperado do número de transições de estado* e o *valor esperado do número de entradas*, respectivamente. O valor esperado do número de transições de estado necessárias para visitar todos os estados de uma MEF pode ser calculado utilizando a fórmula $E(S) = S\ (1 + {}^1/_2 + \ldots + {}^1/_S)$, onde S é o número de estados da MEF. Da mesma forma, o valor esperado do número de entradas necessárias para percorrer todos os caminhos a partir de um dado estado da MEF é calculado através de $E(I) = I$ (1

+ $^1/_2$ + … + $^1/_I$), onde I é o número de entradas da MEF. Entretanto, já que o número de estados necessários para descrever a MEF é desconhecido, S deve ser superestimado *a priori*.

Descrição BNF

O sistema GPK requer uma BNF para a estruturação do cromossomo. A BNF que permite a codificação de cromossomos de tamanho fixo, utilizando representação baseada em estados para 3 entradas, 4 estados e 5 saídas, é definida por:

```
S           ::=   <stat>;
<stat>      ::=   <in_event> <in_event> <in_event> <in_event>;
<in_event>  ::=   <next_st><out> <next_st><out> <next_st><out>;
<next_st>   ::=   "0" | "1" | "2" | "3";
<out>       ::=   "0" | "1" | "2" | "3" | "4";
```

Nesta BNF, o símbolo não terminal *<in_event>* fixa o número de eventos de entrada em três, cada um deles produzindo um par próximo-estado/saída, isto é, *<next_st> <out>*.

Em contraste com o AG clássico, onde os genes são seqüências de bits e a operação de cruzamento pode ser executada em qualquer ponto, GPK somente permite cruzamentos pela troca de sub-árvores a partir de um mesmo símbolo não terminal. A operação de mutação é, por sua vez, implementada como um cruzamento entre o indivíduo selecionado e uma árvore temporária derivada aleatoriamente.

Discussão da Correção Semântica

A priori, a correção semântica, que significa que o modelo implementa corretamente todas as funções desejadas, não pode ser garantida por uma máquina de estados que atenda a seqüências parciais de entrada e saída. No entanto, a metodologia apóia-se no trabalho desenvolvido por Manovit *et al.* [74] e Chongstitvatana e Aporntewan [79] que investigaram a influência do número de seqüências e do comprimento dessas seqüências

no que eles denominaram *porcentagem de correção*. A porcentagem de correção é definida por [74]:

$$Porcentagem\ de\ Correção = \frac{Número\ de\ execuções\ produzindo\ soluções\ completas}{Número\ de\ execuções\ produzindo\ soluções\ incompletas} \quad (4.5)$$

onde *soluções completas* são soluções que operam corretamente para *todas* as possíveis seqüências de entrada e saída, e *soluções incompletas* são soluções que operam corretamente para as seqüências de entrada e saída *testadas*. Os autores mostraram, por meio de experimentos, que a porcentagem de correção pode ser elevada a 100% garantindo, nesse caso, a correção semântica, com o aumento do comprimento das seqüências de treinamento, assim como do número de tais seqüências.

Nos experimentos apresentados na seção 4.7, o comprimento das seqüências de treinamento foi calculado através da equação 4.4 e o número de tais seqüências foi definido a partir de uma tabela fornecida na referência [79]. Tal tabela, obtida experimentalmente, indica quantidades de seqüências de treinamento que garantem porcentagens de correção igual a 100% para circuitos seqüenciais de tamanhos variados.

Nos experimentos apresentados no capítulo 5, o comprimento das seqüências de treinamento foi calculado através da equação 4.4, o que resultou em seqüências de treinamento excessivamente alongadas como conseqüência do grande número de eventos de entrada, que é típico em protocolos de comunicação. Decidiu-se, então, reduzir o comprimento das seqüências de treinamento aumentando a quantidade das mesmas. No experimento de síntese de protocolos, no entanto, o número inicial de seqüências de treinamento pode ser majorado no decorrer do processo evolutivo para satisfazer ao teste de equivalência entre autômatos (Cf. subseções 5.3.3 e 5.3.4).

4.5 Metodologia para a Obtenção de MEFs Mínimas

A abordagem apresentada na seção anterior não otimiza a MEF, deduzida através de processos evolutivos, em relação ao número de estados. A evolução de máquinas de es-

tados que levam em conta o número de estados necessários para implementá-la é de grande interesse, embora pouco tratada pela literatura. A referência [83] denomina essa classe de problemas como *State Adaptive EAs,* isto é, algoritmos evolutivos adaptáveis em relação ao número de estados. Uma vantagem evidente é excluir a utilização de ferramentas de otimização a serem aplicadas em etapas posteriores ao processo evolutivo propriamente dito. Uma outra vantagem em tratar o número de estados durante a evolução do autômato é a possibilidade de aceleração do processo evolutivo. Uma vez que o tamanho do sistema a ser modelado é desconhecido e pode conter menos estados que o esperado, evita-se trabalhar continuamente com codificações extensas de tabelas de transição de estado, como em [78, 79], reduzindo a sobrecarga no processo de avaliação de aptidão.

A presente metodologia emprega MEFs com número variável de estados através de uma modificação na BNF da metodologia básica. A função de aptidão é ponderada pelo número de estados que implementam a MEF, induzindo o processo evolutivo a buscar máquinas de estado que atendam à funcionalidade desejada e que contenham número reduzido de estados. No entanto, como conseqüência do cruzamento entre dois cromossomos de tamanhos diferentes, o processo pode levar a MEFs não realizáveis. Para contornar esta situação, o sistema interpreta estes casos como laços (*self-loops*) reagindo com saída nula.

Função de Aptidão

O número de estados utilizados para implementar a MEF, atribuída a um dado comportamento de entrada e saída, pondera o valor de aptidão. A função de aptidão F assume a forma:

$$F = \left(\sum_{i=1}^{N} w_i H_i \right) * \left(1 + K(S) * GR\right) \tag{4.6}$$

onde N é o número de seqüências de treinamento, w_i é o fator de ponderação para o caso de treinamento i, H_i é o número de acertos relativos ao caso de treinamento i, $K(S)$ é um

fator de ponderação que considera o número de estados *S* utilizados para implementar a MEF e *GR* é o gradiente relativo ao número de acertos do melhor indivíduo. H_i é calculado utilizando os mesmos procedimentos descritos na equação 4.3. *K(S)*GR* aumenta *F* para implementações de MEFs com números menores de estados. Entretanto, quando o valor de aptidão estaciona num ótimo local, *GR* iguala-se a zero criando condições de livre competição entre MEFs, a despeito de seus tamanhos.

Descrição BNF

A seguinte BNF permite a codificação de cromossomos com tamanho variável utilizando representação baseada em estados, considerando 3 entradas, 7 saídas e número máximo de 6 estados.

```
S          ::=   <expr>;
<expr>     ::=   <if_stat> | <expr> <if_stat>;
<if_stat>  ::=   <next_st><out> <next_st><out> <next_st><out>;
<next_st>  ::=   "0" | "1" | "2" | "3" | "4" | "5";
<out>      ::=   "0" | "1" | "2" | "3" | "4" | "5" | "6";
```

Nesta BNF, o símbolo não terminal *<expr>* permite que o cromossomo varie de tamanho, vez que o número de símbolos não terminais *<if_stat>*, que define o número de estados da MEF, pode variar entre indivíduos.

4.6 Metodologia para Tratar a Convergência Prematura

A convergência prematura refere-se a uma situação na qual o processo de otimização de uma função é interrompido antes de encontrar o ótimo global da função. O fenômeno de estagnação em ótimos locais, onde todas as soluções vizinhas não proporcionam melhora, é freqüentemente encontrado em algoritmos evolutivos simples [84], particularmente na evolução de sistemas mais complexos como máquinas de

estados [79]. Esta pausa na evolução de autômatos pode perseverar por centenas ou milhares de gerações em decorrência de uma população "improdutiva".

Uma das principais causas de convergência prematura é a perda de diversidade genética devido à pressão de seleção [84], um problema inerente aos algoritmos evolutivos. Várias técnicas têm sido apresentadas com o objetivo de preservar a diversidade da população no decorrer do processo evolutivo. Chongstitvatana e Aporntewan [79] utilizaram um *rank* de diversidade, ao lado do *rank* de aptidão, no procedimento de seleção de um sistema evolutivo utilizado para gerar circuitos digitais seqüenciais. Dentre as abordagens tradicionais destacam-se as técnicas de *niching*, que se referem à formação de grupos de indivíduos na população. Tais técnicas podem ser classificadas em duas categorias [85]: *crowding* [86] e *fitness sharing* (compartilhamento de recursos) [27].

Na técnica *crowding*, novos indivíduos substituem indivíduos similares, pertencentes a um subconjunto amostrado da população, com menor valor para a função de aptidão. Um problema dessa técnica é sua limitação em otimizar funções multimodais, onde a preservação seletiva de diversidade é crucial. Este fato se deve ao seu potencial *erro de reposição* (*replacement error*), que é definido como a reposição de um membro de um pico (ótimo local) por um membro de outro pico [87]. Mahfold [87] propôs uma variante dessa técnica, que ficou conhecida como "*crowding* determinístico", onde os descendentes competem exclusivamente com seus pais reduzindo sensivelmente o erro de reposição.

Na técnica *fitness sharing*, a aptidão de um indivíduo é reduzida com base no compartilhamento da aptidão entre indivíduos semelhantes. Assume-se, portanto, que há limitação de recursos nos nichos, o que inibe a proliferação de indivíduos semelhantes na população. A figura 4.5 ilustra uma função de compartilhamento, reproduzida de [23], utilizada para definir sobre o compartilhamento de recursos entre dois indivíduos da população.

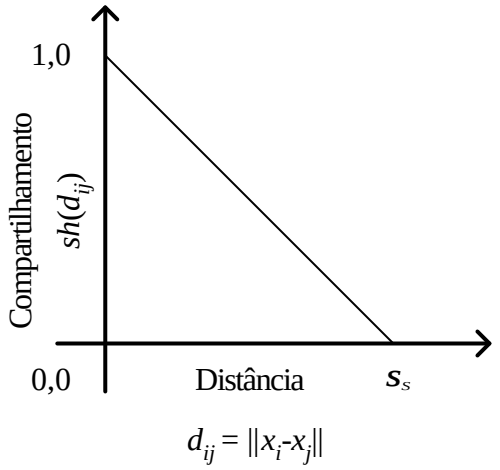


Figura 4.5: Função de compartilhamento triangular.

Observa-se, nesta figura, que indivíduos muito similares a outros indivíduos admitem graus de compartilhamento muito altos, próximos a 1,0, enquanto indivíduos menos similares admitem pequenos graus de compartilhamento, próximos a 0,0. O parâmetro σ_s, denominado raio de compartilhamento (*sharing radius*), estabelece um valor de corte para a distância, isto é, a função $sh(d_{ij})$ assume o valor zero para d_{ij} maior ou igual a σ_s. Na técnica *fitness sharing*, o grau de compartilhamento de recursos de um indivíduo é determinado somando-se os valores da função de compartilhamento contribuídos por todos os indivíduos da população, inclusive pelo próprio. A aptidão compartilhada de um indivíduo, $Fsh(x_i)$, é definida por [88]:

$$Fsh(x_i) = \frac{F(x_i)}{\sum_{j=1}^{M} sh(d(x_i, x_j))} \tag{4.7}$$

onde $F(x_i)$ é a aptidão do indivíduo antes do compartilhamento, M é o número de indivíduos na população, $d(x_i,x_j)$ é uma medida de distância entre dois indivíduos e $sh(d(x_i,x_j))$ é a função de compartilhamento de recursos. Uma desvantagem dessa técnica é seu alto custo computacional. Seu cálculo de aptidão requer uma comparação de cada membro da população com todos os membros da população, isto é, M^2 comparações. A metodologia a ser apresentada nesta seção difere da técnica *fitness*

sharing pelo fato de não haver compartilhamento de recursos e sim um fator de penalidade a ser aplicado a indivíduos com elevada aptidão em circunstâncias de ótimos locais, o que simplifica a implementação e reduz drasticamente o custo computacional.

A metodologia proposta neste trabalho [10] emprega o mesmo fluxograma utilizado pela metodologia básica, apresentada na figura 4.3. No entanto, com o objetivo de auxiliar o algoritmo a evadir-se de um ótimo local, a metodologia inclui uma heurística que direciona o processo de busca no espaço de soluções, doravante denominada *heurística de controle da busca no espaço de soluções*. Para promover a emergência de novas populações, propiciando soluções alternativas, a heurística penaliza o melhor indivíduo e seus variantes toda vez que um ótimo local é detectado, criando condições para o surgimento de novas soluções capazes de exceder o valor de aptidão do ótimo local. A penalidade aplicada às MEFs com elevada aptidão é definida por um fator de penalidade Pf ($0{,}0 \leq Pf < 1{,}0$), e os variantes do melhor indivíduo são determinados mediante um fator de similaridade Sf, definido a seguir.

A similaridade, ou semelhança, entre dois casos, cada um com n atributos, pode ser medida de diferentes maneiras. As medidas de distância calculam a semelhança entre padrões dentro de um espaço geométrico. As mais comuns são a Distância Euclidiana, que mede a distância entre dois casos representados como pontos em um espaço n-dimensional (espaço-n), e a Distância de Hamming, mais básica, freqüentemente utilizada para avaliar o grau de similaridade entre dois vetores binários.

Seja o espaço de Hamming, H^n, definido por:

$$H^n = \{X = (x_1, x_2, ..., x_n)\}, \text{ onde } x_i \in \{1, 0\} \tag{4.8}$$

A Distância de Hamming estabelece o número mínimo de bits que devem ser trocados para converter uma *string* binária em outra. Formalmente, a Distância de Hamming entre duas *strings* ou vetores, X e Y, é definida como:

$$||X, Y|| = \text{número de componentes pelo qual } X \text{ e } Y \text{ diferem} \tag{4.9}$$

Os variantes do melhor indivíduo são determinados por um fator de similaridade baseado no conceito de Distância de Hamming. Para este propósito, a Distância de Hamming clássica, binária, foi estendida para um sistema de notação mais adequado, sendo redefinida como o número de componentes (genes) pelo quais dois vetores (cromossomos) diferem. Dado um cromossomo referência com K-genes, CR_K, o fator de similaridade, Sf, de qualquer outro cromossomo em relação a CR_K é definido por:

$$Sf = \frac{(K - H)}{K} \tag{4.10}$$

onde H é a Distância de Hamming entre eles, em um espaço de Hamming K-dimensional. Por exemplo, o fator de similaridade entre 130212 012001 140120, que descreve CR_K representando uma MEF de três estados, e o cromossomo 130**10**2 0120**23** 14012**2** é Sf = (18-5)/18 = 0,72.

A intenção da heurística de controle da busca no espaço de soluções é eliminar da população corrente os blocos construtores, ou esquemas, que "empurram" os indivíduos em direção ao ótimo local. Pf é ativado quando o gradiente da aptidão do melhor indivíduo, GR, cai abaixo de um determinado limite, caracterizando a presença de um ótimo local. Neste momento, o sistema reduz a taxa dos operadores de decréscimo de diversidade, isto é, cruzamento e clonagem (vide [89]), e eleva a taxa de mutação para aumentar a diversidade da população. Pf permanece ativado por tempo suficiente para "quebrar" os falsos blocos construtores.

Esta metodologia utiliza a função de aptidão definida pela equação 4.3 e uma BNF que permite a codificação de cromossomos de tamanho fixo, semelhante àquela definida na seção 4.4. Na seção 5.5 é apresentado e discutido um exemplo de aplicação da metodologia descrita nesta seção.

4.7 Estudos de Caso: Síntese de Circuitos Lógicos Seqüenciais Síncronos

Nesta seção são descritos experimentos de síntese de circuitos lógicos seqüenciais síncronos de pequeno porte a partir do conhecimento dos eventos de interface dos mesmos. Os circuitos seqüenciais síncronos são circuitos digitais regidos por um sinal de sincronização, denominado relógio (*clock*) do sistema, que define uma cadência para as possíveis alterações no estado dos mesmos. Os seguintes experimentos são tratados nesta seção: (a) contador módulo-4 (subseção 4.7.1); (b) detector de seqüência de 4 bits (“1011”) (subseção 4.7.2); e (c) somador binário seqüencial (subseção 4.7.3).

4.7.1 Contador Módulo-4

Um contador módulo-*p*, ou contador divisor-por-*p* (*divide-by-p counter*), é um sistema seqüencial cuja entrada é uma variável binária e cuja saída apresenta valores inteiros do conjunto {0, 1, ..., *p*-1}. Uma descrição de estados do contador módulo-*p* requer, no mínimo, *p* estados. Caso a descrição seja feita com número mínimo de estados, estes são dispostos em ciclo único.

O contador módulo-4 a ser evoluído sinaliza com saída alta (nível lógico “1”) durante um pulso de relógio (voltando a “0” em seguida) sempre que ocorrer uma seqüência de quatro “1s” em sua entrada, consecutivos ou não. O objetivo do experimento é gerar um modelo em máquinas de estados finitos com número mínimo de estados a partir de seqüências de entrada e saída corretas fornecidas na entrada do sistema.

Foram realizados experimentos com uma população de 500 MEFs que evoluiu até um máximo de 3000 gerações. A taxa de cruzamento e a taxa de mutação foram estabelecidas em p_c = 0,65 e p_m = 0,05, respectivamente. Utilizou-se o método de seleção *Linear Rank* com elitismo. A adoção da estratégia do elitismo neste experimento, diferentemente da abordagem utilizada no capítulo 3 (sem elitismo), deveu-se ao fato de que tal estratégia acelera o processo evolutivo em geral. A população foi moldada utilizando a BNF descrita na seção 4.5, permitindo a produção de MEFs com até 6 estados. O fator

$K(S)$ foi definido pelo conjunto de valores $K(S)$ = {0,01, 0,008, 0,006, 0,004, 0,002, 0,000} para S = {1, 2, 3, 4, 5, 6}, respectivamente.

As seqüências de entrada foram geradas aleatoriamente e as seqüências de saída foram produzidas pela aplicação das seqüências de entrada na entrada da MEF a ser modelada, após esta ter sido colocada no estado inicial. O comprimento das seqüências de treinamento foi calculado utilizando-se 2 eventos de entrada (I), "0" e "1", e o número de estados (S) foi estimado em 5, o que resultou em seqüências de treinamento com 36 pares de eventos de entrada e saída (Cf. equação 4.4). Ao todo, foram utilizadas N = 8 seqüências de treinamento de tamanhos iguais (w_i = 1, nas equações 4.3 e 4.6), produzindo um máximo de 8 × 36 = 288 acertos. Os resultados, apresentados na tabela 4.1, foram obtidos após 20 execuções independentes em três diferentes configurações do sistema, quais sejam:

- Configuração 1: A função de aptidão não leva em consideração o número de estados da MEF. Usa-se, portanto, a equação 4.3;
- Configuração 2: A função de aptidão é ponderada pelo número de estados da MEF, de acordo com a equação 4.6. Nesse caso, GR é calculado continuamente sobre as últimas 50 gerações, sendo que nas 50 primeiras, assim como em gerações posteriores ao surgimento de MEFs funcionalmente corretas, isto é, com número máximo de acertos, seu valor é fixado em 1; p_m é alterada para 0,3 sempre que $GR \leq 0{,}1$;
- Configuração 3: A função de aptidão é dividida em duas fases: a primeira fase utiliza a equação 4.3 e termina quando o primeiro indivíduo alcança o número máximo de acertos; a segunda fase é ativada em seguida e utiliza a equação 4.6 com o objetivo de buscar MEFs corretas porém reduzidas, isto é, otimizadas em relação ao número de estados.

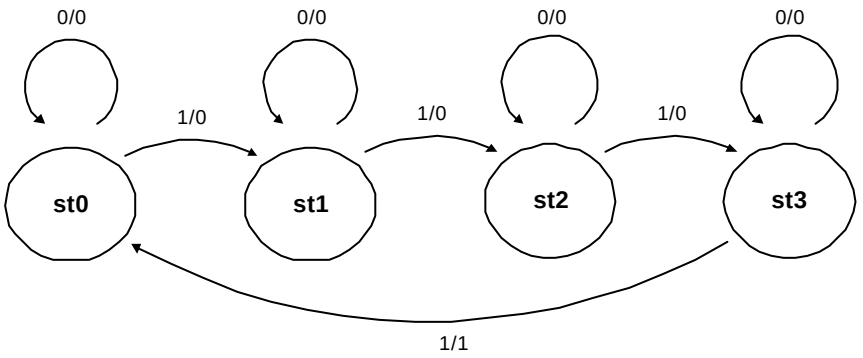
Tabela 4.1: Evolução do contador módulo-4: comparação entre três configurações do sistema (20 execuções).

Configuração	Execuções que não convergiram para G_{MAX} = 3000	Execuções que produziram soluções funcionalmente corretas para G_{MAX} = 3000

		4 estados	5 estados	6 estados
1. Equação 4.3	2	2	3	13
2. Equação 4.6	3	9	6	2
3. 1ª Fase: Eq. 4.3 2ª Fase: Eq. 4.6	2	4	7	7

A tabela 4.1 revela que a ausência de restrições na função de aptidão, adotada na configuração 1, resulta, na maioria das vezes, em soluções com elevado número de estados, produzindo soluções otimizadas (MFEs funcionalmente corretas com número mínimo de estado) em apenas 10% das execuções. A configuração 2, por sua vez, produziu soluções otimizadas em 45% das execuções. Esta configuração apresentou melhora de 25% em relação à configuração 3, que implementa o método clássico de se separar o processo de busca em duas fases (vide [90]), uma para cada objetivo, indicando que, em se tratando de sistemas seqüenciais, é importante restringir o número de estados que implementam a MEF durante a etapa principal da evolução.

A figura 4.6 mostra a máquina Mealy resultante que descreve com sucesso o contador módulo-4 utilizando um grafo de transição de estado (STG: *State-Transition Graph*). Nesta figura, um rótulo *v*/*v'* representa um arco, e_{ij}, entre dois estados, q_i e q_j, se, e somente se, $O(q_i,v) = q_j$ e $T(q_i,v) = v'$. Esta MEF alcançou o total de 288 acertos utilizando o número mínimo de 4 estados.



Cromossomo: 0010 1020 2030 3001

Figura 4.6: STG do contador módulo-4.

4.7.2 Detector de Seqüência de 4 Bits (“1011”)

Um detector de seqüência, ou reconhecedor de padrão [81], é um sistema seqüencial de memória finita cuja saída binária indica se a subseqüência de entrada corresponde ao padrão particular reconhecido pelo sistema. Assume-se, no presente experimento, que a máquina de estados a ser modelada não deverá detectar padrões sobrepostos.

O detector de seqüência de 4 bits “1011” é uma máquina seqüencial que sinaliza com saída alta (nível lógico “1”) durante um pulso de relógio (voltando a “0” em seguida) sempre que ocorrer a presença do trem de pulsos “1011” na entrada do circuito. Da mesma forma que na subseção anterior, o objetivo deste experimento é gerar um modelo em máquinas de estados finitos com número mínimo de estados a partir de seqüências de entrada e saída corretas fornecidas na entrada do sistema.

Os parâmetros do sistema de PG são os mesmos adotados na subseção anterior, com exceção do número máximo de gerações, estabelecido em 2000. As seqüências de entrada foram geradas aleatoriamente e as seqüências de saída foram produzidas pela aplicação daquelas na entrada da MEF a ser modelada, após esta ter sido colocada no estado inicial. O comprimento e a quantidade de seqüências de treinamento também se repetiram, visto que ambos os problemas apresentam o mesmo número de entradas e que o número de estados necessários à modelagem é desconhecido. Os resultados, apresentados na tabela 4.2, foram obtidos após 20 execuções independentes nas três configurações do sistema apresentadas na subseção anterior.

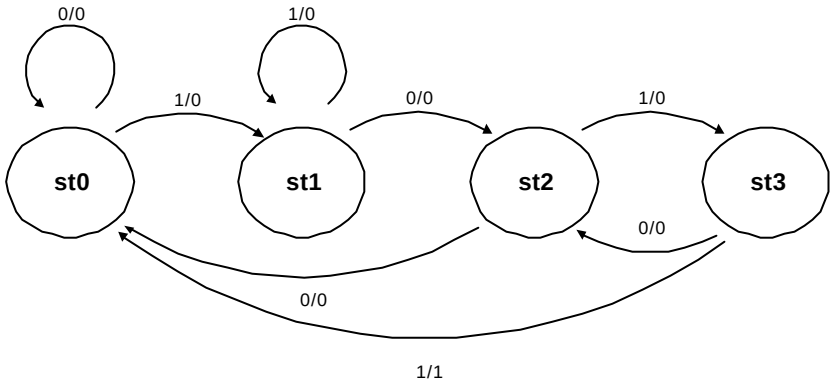
Tabela 4.2: Evolução do detector de seqüência de 4 bits: comparação entre três configurações do sistema (20 execuções).

Configuração	Execuções que não convergiram para G_{MAX} = 2000	Execuções que produziram soluções funcionalmente corretas para G_{MAX} = 2000		
		4 estados	5 estados	6 estados
1. Equação 4.3	5	2	2	11

2. Equação 4.6	6	12	2	-
3. 1ª Fase: Eq. 4.3 2ª Fase: Eq. 4.6	5	11	4	-

Apesar do detector de seqüência apresentar maior dificuldade de evolução em relação ao contador módulo-4, por haver menor relação entre "1s" e "0s" nas saídas das seqüências de treinamento, os resultados apresentados na tabela 4.2 confirmam as conclusões obtidas no experimento anterior, embora com menor margem de vantagem da configuração 2 sobre a configuração 3, mostrando a eficácia da técnica de obtenção de MEFs com número mínimo de estados (uso da equação 4.6).

A figura 4.7 mostra a máquina Mealy resultante que descreve com sucesso o detector de seqüência "1011", utilizando um grafo de transição de estado (STG). Esta MEF alcançou o total de 18 × 32 = 288 acertos utilizando o número mínimo de 4 estados.



Cromossomo: 0010 2010 0030 2001

Figura 4.7: STG do detector de seqüência "1011".

4.7.3 Somador Binário Seqüencial

Uma célula do circuito conhecido como "somador completo" (*full adder*) implementa um somador de 1 bit que produz dois sinais de saída, isto é, a saída propriamente dita e uma saída auxiliar indicando a sinalização "vai um" (*carry*), a partir de três sinais de

entrada, ou seja, os dois sinais a serem somados e o "vai um" da operação de soma anterior. O somador binário seqüencial implementa um somador de 1 bit, serial, gerando na saída a soma de duas seqüências de entrada. A figura 4.8 mostra a composição funcional do somador binário seqüencial a ser evoluído. Os sinais observáveis são, portanto, os dois sinais de entrada e o sinal de saída com o resultado da operação de adição desses dois sinais. O circuito correto deve considerar automaticamente o "vai um" do resultado do último par de bits somado, sendo, portanto, classificado como circuito seqüencial.

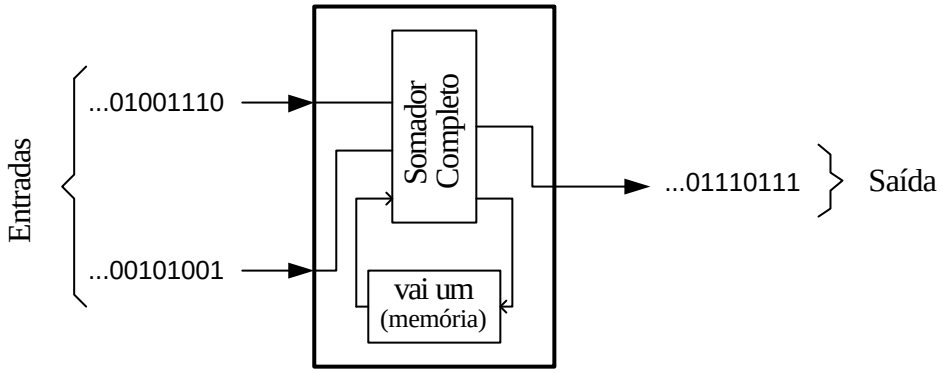


Figura 4.8: Somador binário seqüencial.

Deseja-se gerar uma MEF que modele o somador binário seqüencial, com número mínimo de estados, pela observação de seus eventos de interface. Para tanto, foram realizados experimentos com uma população de 200 MEFs que evoluiu até um máximo de 1000 gerações. A taxa de cruzamento e a taxa de mutação foram estabelecidas em $p_c = 0{,}65$ e $p_m = 0{,}05$, respectivamente. Utilizou-se o método de seleção *Linear Rank* com elitismo. A população foi moldada utilizando a BNF descrita na seção 4.5, permitindo a produção de MEFs com até 4 estados.

O comprimento das seqüências de treinamento foi calculado utilizando-se 4 eventos de entrada (I), $V = \{00, 01, 10, 11\}$, e número de estados (S) estimado em 3, o que resultou em seqüências de treinamento com 48 pares de eventos de entrada e saída (Cf. equação 4.4). Foram utilizadas $N = 8$ seqüências de 48 bits, geradas a partir de uma distribuição uniforme entre os elementos de V, produzindo um máximo de $8 \times 48 = 384$ acertos. Os resultados, apresentados na tabela 4.3, foram obtidos após 20 execuções independentes nas três configurações do sistema apresentadas na subseção 4.7.1.

A evolução do somador binário seqüencial mostrou-se mais comportada que a do detector de seqüência, discutido na subseção anterior, por apresentar uma distribuição uniforme entre "0s" e "1s" na saída. Mais uma vez comprovou-se a eficácia da técnica de obtenção de MEFs com número mínimo de estados a partir do uso da equação 4.6.

Tabela 4.3: Evolução do somador binário seqüencial: comparação entre três configurações do sistema (20 execuções).

Configuração	Execuções que não convergiram para G_{MAX} = 1000	Execuções que produziram soluções funcionalmente corretas para G_{MAX} = 1000		
		2 estados	3 estados	4 estados
1. Equação 4.3	3	1	5	11
2. Equação 4.6	7	11	1	1
3. 1ª Fase: Eq. 4.3 2ª Fase: Eq. 4.6	3	5	11	1

A figura 4.9 mostra a máquina Mealy resultante que descreve com sucesso o somador binário seqüencial utilizando um grafo de transição de estado (STG). Esta MEF alcançou o total de 384 acertos utilizando apenas 2 estados.

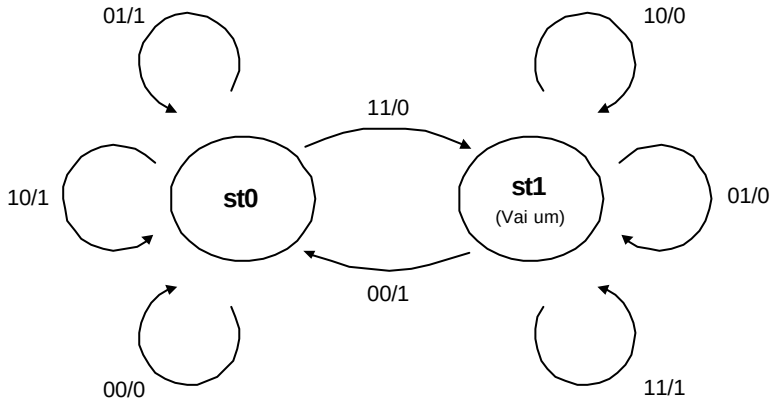


Figura 4.9: STG do somador binário seqüencial.

A metodologia para tratar a convergência prematura, apresentada na seção 4.6, não foi aplicada nos estudos de caso apresentados neste capítulo devido à simplicidade desses experimentos. No entanto, ela será aplicada em um problema cuja solução requer MEFs mais complexas, a ser apresentado no próximo capítulo (seção 5.5).

4.8 Comentários

Neste capítulo foram apresentadas metodologias de síntese de sistemas seqüenciais a partir de seqüências parciais de entrada e saída através de técnicas evolutivas, com emprego de máquinas de estados finitos. Os métodos convencionais de síntese de sistemas seqüenciais requerem o conhecimento prévio do comportamento do circuito na forma de um diagrama de estados, impossibilitando o tratamento de problemas deste tipo. Por outro lado, a grande maioria das aplicações de técnicas evolutivas utiliza *aprendizado supervisionado* [3], isto é, a evolução é impulsionada por seqüências de treinamento compostas por entradas e saídas desejadas, o que as torna adequadas para tratar problemas desta natureza.

Nas metodologias aqui apresentadas, a correção semântica do modelo de máquina de estados finitos gerado está implícita no processo evolutivo. A justificativa para essa suposição baseia-se na escolha adequada da quantidade de seqüências de treinamento e de seus comprimentos, como sugerido no método proposto em [74, 79], discutido na seção 4.4, e confirmado por experimentos realizados naqueles trabalhos e no presente capítulo.

Os estudos de caso demonstraram a potencialidade das metodologias em modelar sistemas onde se tem acesso apenas a eventos de entrada e saída. Foram comprovados os resultados de experimentos de circuitos lógicos seqüenciais apresentados em [79], tais como o contador módulo-4 e o somador binário seqüencial. No entanto, diferentemente daquele trabalho, a metodologia aqui desenvolvida permitiu fornecer o modelo de MEF com o número de estados já minimizado.

Este capítulo apresentou também uma metodologia para tratar o problema da convergência prematura em sistemas evolutivos seqüenciais. A heurística nela

empregada para controlar o processo de busca no espaço de soluções é de fácil implementação e apresenta baixo custo computacional se comparada a outras técnicas encontradas na literatura. Assim como na técnica *crowding*, a metodologia proposta apresenta limitações para otimizar funções multimodais. No entanto, para problemas tratados neste trabalho, tal como a evolução de máquinas de estados, a técnica apresentada mostrou-se bastante eficaz (vide estudo de caso na seção 5.5).

As metodologias propostas neste capítulo, essencialmente direcionadas a aplicações de engenharia reversa, podem também ser utilizadas em abordagens de engenharia direta. No capítulo 5 (seção 5.5) será apresentado um estudo de caso que utiliza uma dessas metodologias para gerar modelos para entidades de protocolo a partir de cenários de comunicações.

As metodologias apresentadas neste capítulo são estendidas no capítulo a seguir para tratar projeto de protocolos de comunicação tanto em abordagens sintéticas quanto em abordagens analíticas.

Capítulo 5

Projeto de Protocolos de Comunicação

5.1 Introdução

Nas últimas décadas, os sistemas distribuídos têm se consolidado como espinha dorsal de aplicações baseadas em sistemas informatizados, tais como sistemas bancários, bancos de informação e, principalmente, *internet*. Nesses sistemas residem os protocolos de comunicação, sistemas estes que coordenam a troca de mensagens e sinalização entre entidades computacionais autônomas, fisicamente dispersas, com o objetivo de prover aos usuários os serviços especificados pelas aplicações.

Um protocolo de comunicação pode ser entendido como um sistema reativo, modelado por uma máquina de estados finitos, dominado, essencialmente, por fluxo de controle. Seu projeto tem sido objeto de interesse de diversos grupos de pesquisa desde a década de 70, período de disseminação das redes de comunicação de dados. Apesar de várias metodologias propostas neste período entre abordagens sintéticas e analíticas, o projeto de protocolos continua sendo uma tarefa difícil e que tende a gerar erros [91].

Diferentemente das metodologias encontradas na literatura, essencialmente de natureza determinística, este capítulo apresenta duas metodologias de projetos de protocolos de comunicação fundamentadas em técnicas evolutivas. Essas metodologias são baseadas nas metodologias descritas no capítulo 4, supondo, portanto, que os protocolos apresentem comportamento seqüencial. Após uma introdução aos sistemas de eventos discretos na seção 5.2, são apresentadas a primeira metodologia, na seção 5.3, que aborda a síntese de protocolos, e a segunda, na seção 5.4, que trata o projeto de protocolos a partir de cenários de interação. São apresentados também os resultados de dois experimentos utilizados para testar a viabilidade e o desempenho dessas metodologias. Por último, a seção 5.5 apresenta e analisa os resultados obtidos com a aplicação da metodologia de

tratamento de convergência prematura, descrita no capítulo 4 (seção 4.6), no problema tratado na subseção 5.4.5.

5.2 Sistemas de Eventos Discretos

Sistemas de eventos discretos (DES: *Discret Event Systems*) são sistemas de processos cooperantes que são discretos, assíncronos e, possivelmente, não determinísticos [92]. Nesses sistemas, os eventos, que podem corresponder à recepção ou transmissão de uma mensagem, são executados instantaneamente causando uma mudança discreta no estado do sistema. O projeto de sistemas de eventos discretos é fundamentado, principalmente, em duas abordagens [92, 93]: análise e síntese.

A abordagem de análise, ou abordagem analítica, parte de uma especificação informal e o sistema é construído de maneira incremental, não estruturada. Esta abordagem resulta, invariavelmente, em projetos incompletos e errôneos. Por esta razão, o sistema deve ser testado, com auxílio de ferramentas de verificação, após cada refinamento. A seqüência de re-projeto, análise, detecção e correção de erros é aplicada iterativamente até que o sistema se torne livre de erros. Uma das principais desvantagens da abordagem de análise em projetos de sistemas discretos está na dificuldade de reformulação ou alteração de um projeto existente.

A abordagem de síntese, ou abordagem sintética, é baseada na geração de um projeto a partir de uma especificação formal em alto nível. O projeto parte de requerimentos em alto nível, na forma de uma especificação formal, que são automaticamente ou semi-automaticamente refinados a fim de obter o detalhamento do sistema. Um aspecto importante da abordagem de síntese é que ela deve garantir por si só a correção sintática e semântica do sistema gerado. O processo de síntese beneficia-se de ferramentas de CAD, encurtando o ciclo de desenvolvimento do projeto e facilitando alterações em sistemas já existentes.

Os *protocolos de comunicação* fazem parte de uma classe de sistemas de eventos discretos de interesse particular deste trabalho. Um protocolo de comunicação é definido como um conjunto de regras que governa o formato e o significado de quadros, paco-

tes ou mensagens que são trocados entre entidades pares (computadores) [94]. Os protocolos de comunicação são componentes fundamentais dos *sistemas distribuídos*. Sistemas distribuídos são aqueles em que diversos computadores interconectados, formando uma ou mais redes de computadores, trabalham coordenadamente em processos que envolvam informações ou recursos remotos de tal modo que a distribuição de tarefas e informações não se torne aparente ao usuário. Inúmeras pesquisas têm abordado o projeto de sistemas distribuídos, em grande parte através de síntese, tais como [92, 95, 96, 97, 98, 99].

Os sistemas distribuídos são especificados segundo dois níveis de abstração [100]. No nível de *especificação de serviço*, as entidades de protocolo e os canais de comunicação, que ligam tais entidades, estão escondidos. A especificação nesse nível é descrita exclusivamente através de trocas de *primitivas de serviços* em pontos de acesso ao serviço (SAPs: *Service Access Points*). No nível de *especificação de protocolo*, para cada entidade de protocolo, suas próprias ações, as mensagens trocadas com outras entidades de protocolo, e a correta ordenação dessas mensagens são descritas. A especificação de serviço e a especificação de protocolo devem ser equivalentes, ou seja, devem exibir o mesmo comportamento aos usuários do serviço.

Independentemente da abordagem utilizada, análise ou síntese, dois tipos de propriedades devem ser garantidos no projeto de protocolos de comunicação [92]: segurança (*safety*) e vivacidade (*liveness*). As propriedades de segurança dizem respeito à correção sintática, ou lógica, garantindo que o protocolo não entrará em estado ou situação indesejados. Elas incluem ausência de *deadlock*, ausência de *livelock* e completude (*completeness*), isto é, ausência de erros devido à recepção de evento não especificado.

As propriedades de vivacidade dizem respeito à correção semântica, assegurando que o protocolo provê as funções a serem fornecidas aos usuários especificadas na descrição do serviço do protocolo. A referência [92] menciona um terceiro tipo de propriedade relativo à capacidade de resposta ou sensibilidade (*responsiveness*) do protocolo, como por exemplo, a tolerância do sistema a falhas. Por serem consideradas de caráter específico, as propriedades de resposta não são tratadas neste trabalho.

5.3 Síntese de Protocolos de Comunicação

A *síntese de protocolo* é uma importante etapa do ciclo de engenharia de protocolo, ilustrado na figura 5.1. Ela pode ser definida como a geração de especificações de protocolo corretas a partir de especificações de serviço do protocolo [100]. Esta definição assegura que o processo de síntese não requer uma futura validação sintática e semântica [91]. A etapa de implementação encerra o ciclo de projeto do protocolo gerando um produto em *software*, *hardware*, ou uma combinação de ambos, através da técnica conhecida como *Hardware*/*Software Codesign* [28].

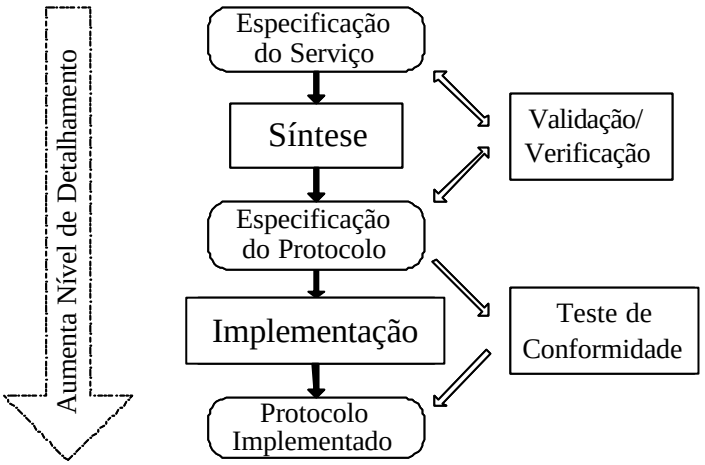


Figura 5.1: Ciclo de engenharia de protocolo.

Geralmente, o processo de síntese utiliza o mesmo formalismo empregado na especificação de serviço do protocolo para gerar a especificação do protocolo. No entanto, além do modelo do serviço de protocolo, que estabelece a troca de informação entre o usuário e o provedor do serviço, é necessário fornecer ao sistema de síntese um conjunto de mensagens a serem trocadas entre as entidades de protocolo.

5.3.1 Trabalhos Relacionados

Várias estratégias têm sido propostas com o objetivo de gerar automaticamente es-

pecificações de protocolo a partir de especificações de serviço do protocolo, principalmente para modelos de MEF, LOTOS, Redes de Petri e suas extensões. A referência [93] fornece uma revisão completa dos métodos de síntese de protocolos existentes até sua data de publicação. Mais recentemente, como destacado em [101], estas estratégias têm-se preocupado, principalmente, em implementar fluxos de controle complexos [97], suportar restrições de tempo [98, 99] e gerenciar recursos distribuídos [102].

Yamaguchi *et al*. [97] propuseram um algoritmo que sintetiza automaticamente a especificação da entidade de protocolo a partir da especificação do serviço do protocolo e de um conjunto de portas lógicas e registradores em um modelo de Redes de Petri com Registradores (PNR: *Petri Net model with Registers*). A idéia básica dos autores foi substituir cada transição da especificação do serviço por uma sub-rede de Petri que simulasse a transição. Esta abordagem permitiu descrever fluxos de controle estruturados, como por exemplo, aqueles contendo eventos paralelos.

Park e Miller [98] propuseram um modelo de especificação denominado *Time Extended* FSM (TEFSM) e apresentaram um algoritmo para gerar especificações de protocolo a partir de especificações de serviço do protocolo utilizando este modelo. TEFSM permite descrever explicitamente eventos concorrentes, sincronização e restrições de tempo tais como atraso e estouro de contador (*time-out*).

El-Fakih *et al*. [102] propuseram um método para a síntese de protocolos de comunicação levando em conta a otimização de mensagens trocadas entre entidades de protocolo. Este método utilizou um AG com a função específica de otimizar o número de mensagens em trânsito considerando uma quantidade fixa de recursos disponíveis em cada entidade de protocolo.

5.3.2 Especificação do Serviço do Protocolo

Esta subseção exemplifica uma especificação simplificada do serviço fornecido por um protocolo de comunicação orientado à conexão. O serviço a ser fornecido pelo protocolo é especificado por um conjunto de *primitivas de serviço* disponível ao usuário para este acessar o serviço. Esta especificação utiliza o modelo de MEF e está apresentada na figura 5.2. A figura 5.2a fornece a especificação global do serviço do protocolo

orientado à conexão, denotado por ES_G. As figuras 5.2b e 5.2c fornecem a especificação do serviço (*ES*) segundo a perspectiva da entidade emissora (ES_E) ou receptora (ES_R). Nestas figuras, uma primitiva A_i ocorre no *site i*; um envio (para o usuário) e uma recepção (do usuário) da primitiva A_i são denotados $!A_i$ e $?A_i$, respectivamente. Estando as entidades em repouso (estado 1), a conexão é iniciada pelo emissor (*C_Req*) e o receptor pode aceitá-la (*C_Resp*) ou rejeitá-la (*D_Req*). Caso o serviço esteja estabelecido (estado 5 na figura 5.2a e estado 3 nas figuras 5.2b e 5.2c), o emissor pode iniciar o procedimento de reinicialização (*R_Req*) para remover todas as mensagens do meio de comunicação. Finalmente, a conexão pode ser liberada pelo emissor, ou pelo receptor, através da primitiva de serviço *D_Req*. No modelo de serviço do protocolo apresentado na figura 5.2, a fase de transferência de dados foi omitida. Tal omissão constitui uma prática freqüentemente adotada em abordagens de síntese de protocolos [93].

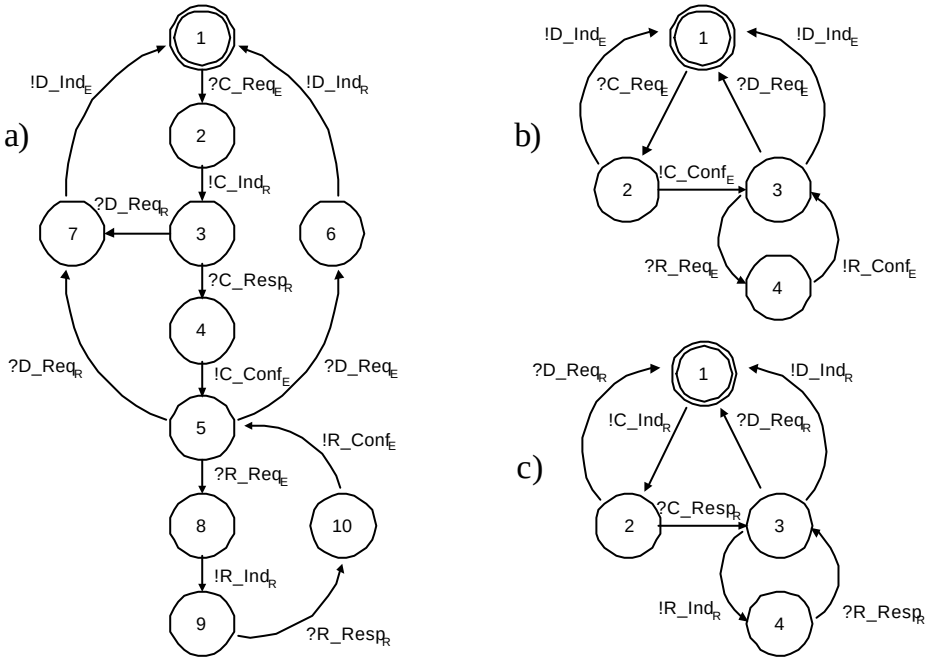


Figura 5.2: (a) *ES* global (ES_G); (b) *ES* do emissor (ES_E); e (c) *ES* do receptor (ES_R).

5.3.3 Cálculo de Processos: Descrição Formal com CCS e Verificação

O desenvolvimento de sistemas de computação distribuídos complexos requer sofisticadas técnicas de verificação para garantir sua correção. A contínua elevação do nível de detalhamento de tais sistemas os torna intratáveis do ponto de vista manual, demandando o auxílio de ferramentas de CAD. Sistemas de estados finitos, tais como protocolos de comunicação e *hardware*, são particularmente adequados para a análise automática devido a sua natureza finita, o que garante a existência de procedimentos de decisão capazes de revelar importantes propriedades desses sistemas [103].

Nessa subseção são apresentadas, sucintamente, uma técnica de descrição formal (TDF) de sistemas distribuídos, o CCS (*Calculus of Communicating Systems*) [104], e uma ferramenta que realiza o teste de equivalência entre o modelo do protocolo e o modelo do serviço do protocolo especificados segundo esta TDF, o CWB (*Concurrency Workbench*) [103], a serem utilizadas na metodologia de síntese de protocolos de comunicação apresentada na seção 5.3. Detalhes podem ser consultados no apêndice A.

O cálculo de processos provê uma abordagem algébrica para a modelagem e o raciocínio sobre sistemas reativos. Ele pode ser entendido como uma base matemática para a definição de linguagens de programação. A abstração por ele proporcionada, no entanto, é direcionada à análise de diversos fenômenos semânticos, tal como não-determinismo. O cálculo de processos ignora, intencionalmente, conceitos importantes de linguagens de programação convencionais, tal como estrutura de dados, e concentra-se na modelagem do comportamento da comunicação entre sistemas concorrentes, provendo uma base para métodos de análise e verificação.

O CCS é uma TDF baseada no cálculo de processos, aplicada ao estudo de processos concorrentes e de protocolos em particular. Este formalismo define um rigoroso conjunto de regras de transformação e relações de equivalência que permitem ao projetista raciocinar formalmente sobre comportamentos [91]. Como uma álgebra, CCS provê um conjunto de termos, operadores e axiomas que podem ser utilizados para escrever e manipular expressões algébricas. As expressões algébricas definem os elementos de um sistema concorrente e a manipulação dessas expressões revela como o sistema se comporta.

CCS provê fundamentos para responder questões pragmáticas, como por exemplo, a determinação da igualdade entre dois agentes em termos de comportamento. Se dois

processos apresentam diferentes implementações, então pode-se determinar se um deles pode ser substituído pelo outro. Com CCS, sistemas podem ser comparados utilizando a noção de *equivalência observacional*. Dois termos, *M* e *N*, são observacionalmente equivalentes se, e somente se, para todo contexto *C*[], onde *C*[*M*] é um termo válido, *C*[*N*] é também um termo válido com o mesmo valor. Nesse caso, sempre que um deles realizar uma transição observável o outro pode realizar a mesma transição observável e evoluir para um estado equivalente.

O elemento básico do CCS é um agente que exibe um comportamento único. Esse comportamento é definido por um conjunto de eventos ou ações que o agente é capaz de realizar. Por exemplo, um agente BUFFER pode ser usado para relacionar um conjunto de eventos, tal como, {*in*, *out*}. A seqüência dos eventos por ele realizada é descrita por meio de um prefixo de ação, denotado por um "• ". No exemplo abaixo, o agente BUFFER é definido para realizar a seqüência de ações *in* e *out*, ou seja, para armazenar um valor e, em seguida, recuperar o mesmo valor. Formalmente, tem-se:

$$Agent\ BUFFER = in.'out.BUFFER \tag{5.1}$$

Em CCS, um sistema reativo é criado através do operador de composição, representado por uma barra vertical. Na definição abaixo, o sistema reativo *PQ* é definido como uma composição dos agentes *P* e *Q* que atuam independentemente, mas que podem interagir através da ação *c*.

$$agent\ P = a.'c.P \tag{5.2}$$

$$agent\ Q = c.'b.Q \tag{5.3}$$

$$agent\ PQ = (P|Q) \tag{5.4}$$

Neste exemplo, o evento (ou ação) de entrada *c* e o evento de saída *'c* são ditos complementares, refletindo o fato de que eles representam entrada e saída em uma mesma porta ou "rótulo" (*c*, no caso). A figura 5.3 mostra as interações internas e externas no sistema reativo *PQ*, baseadas nas equações 5.2, 5.3 e 5.4.

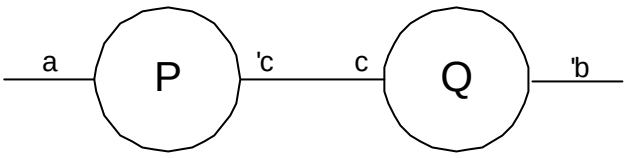


Figura 5.3: Sistema reativo *PQ*: interações internas e externas.

O CWB é uma ferramenta de análise de sistemas concorrentes, tendo sido aplicado com sucesso na verificação de protocolos de comunicação, notavelmente dos protocolos Bit Alternante e CSMA/CD [105], e de algoritmos de exclusão mútua [103]. Atualmente, há duas versões do CWB que apresentam comandos diferentes para os mesmos propósitos: a versão "Edinburgh" e a versão "North Carolina", desenvolvidas pela Universidade de Edinburgh e pela Universidade do Estado da Carolina do Norte, respectivamente. Apesar de capacitado a realizar várias tarefas, como a verificação de erros lógicos, o CWB foi utilizado neste trabalho exclusivamente para a verificação de equivalência entre sistemas especificados com diferentes níveis de detalhamento, na metodologia de síntese de protocolos de comunicação apresentada a seguir.

5.3.4 Metodologia

Dentre os modelos formais utilizados para especificar protocolos de comunicação, o modelo de máquinas de estados finitos (MEFs) é o mais empregado devido à sua simplicidade e definição precisa da ordenação temporal das interações [92, 106, 107]. A presente metodologia gera as especificações das entidades de protocolo a partir da especificação do serviço do protocolo utilizando como formalismo as MEFs.

A metodologia de síntese de protocolos de comunicação [11, 12] é uma extensão das metodologias de síntese de sistemas seqüenciais a partir de seqüências parciais de entrada e saída apresentadas no capítulo 4. O protocolo a ser sintetizado é, portanto, assumido de natureza seqüencial, tomando-se o cuidado para que esta simplificação não comprometa suas funções essenciais [99]. A abordagem tem seu foco na parte de controle

do protocolo, excluindo aspectos de dados, como por exemplo, mensagens com parâmetros. Tal restrição é adotada na maioria das abordagens de síntese de protocolos [93].

Em vez de criar uma especificação global de estados do protocolo e projetá-la no alfabeto de eventos de cada *site* (entidade), como em [92, 99], a metodologia gera as entidades de protocolo separadamente. As etapas envolvidas neste procedimento são semelhantes àquelas discutidas na metodologia para a obtenção de MEFs mínimas, apresentada na seção 4.5. No entanto, a presente metodologia propõe uma heurística para gerar seqüências de treinamento a partir da especificação do serviço do protocolo e de um conjunto de mensagens ou *unidades de dados de protocolo* (PDUs: *Protocol Data Units*) a ele associado, e checa, ao final da evolução das MEFs, a equivalência entre a especificação do serviço do protocolo, fornecida na entrada, e os modelos das entidades de protocolo evoluídos. A heurística para a obtenção das seqüências de treinamento é detalhada na subseção 5.3.5.

A figura 5.4 mostra o fluxograma completo da metodologia de síntese de protocolos de comunicação. Nesse fluxograma, as entidades pares são evoluídas separadamente até atenderem às suas respectivas restrições de entrada e saída. Inicialmente, a quantidade e o comprimento das seqüências de treinamento são estabelecidos de acordo com o procedimento descrito na seção 4.4. No momento em que as restrições dadas pelas seqüências de treinamento são satisfeitas, as MFEs das entidades pares são convertidas para CCS com o objetivo de realizar o teste de equivalência entre o modelo do protocolo e o modelo do serviço do protocolo, que também é convertido para CCS. Em caso positivo, a metodologia finda apresentando as MEFs das entidades de protocolo corretas. Em caso negativo, o número de seqüências de treinamento dos sistemas evolutivos (um para cada entidade) é incrementado em uma unidade e os algoritmos evolutivos buscam novamente por MEFs que atendam às novas restrições de entrada e saída para serem submetidas, ao final da evolução, a um novo teste de equivalência. É importante observar que os sistemas evolutivos não são interrompidos após o incremento do número de seqüências de treinamento, que modificam o cálculo das funções de aptidão, livrando-os do ônus de uma reinicialização.

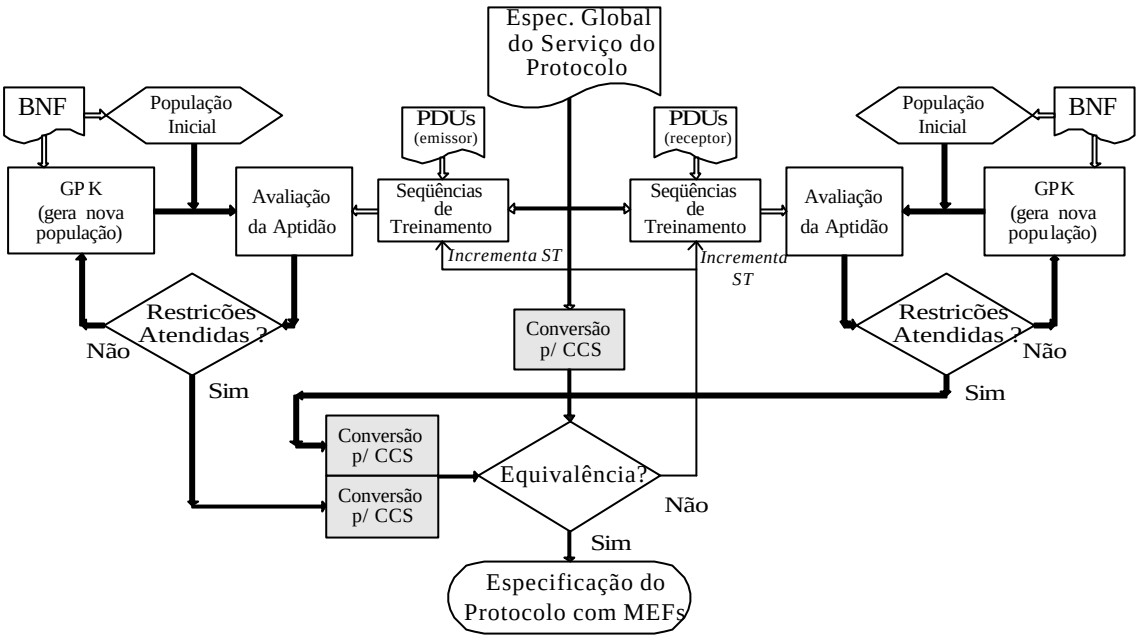


Figura 5.4: Metodologia de síntese de protocolos de comunicação.

O modelo adotado para o cálculo de aptidão está esboçado na figura 5.5. Este modelo é similar à abordagem de "caixa preta" utilizada em testes de conformidade de protocolos de comunicação [106]. Utilizando uma interface estimulada por eventos, a *entidade a ser evoluída* comunica-se com o usuário do serviço através de primitivas de serviço e com a entidade par através de PDUs. Neste modelo é assumido que o meio de comunicação é confiável, não havendo, portanto, ocorrência de falhas na rede. Essa restrição também é adotada na maioria das abordagens de síntese de protocolos [93]. Em cada geração, o sistema estimula uma população de MEFs com seqüências de entrada particulares (canais IN_0 e IN_1) e, em seguida, armazena as seqüências de saída correspondentes (canais OUT_0 e OUT_1). Estas seqüências de saída são comparadas às saídas corretas (ou desejadas) e um valor de aptidão é atribuído a cada indivíduo.

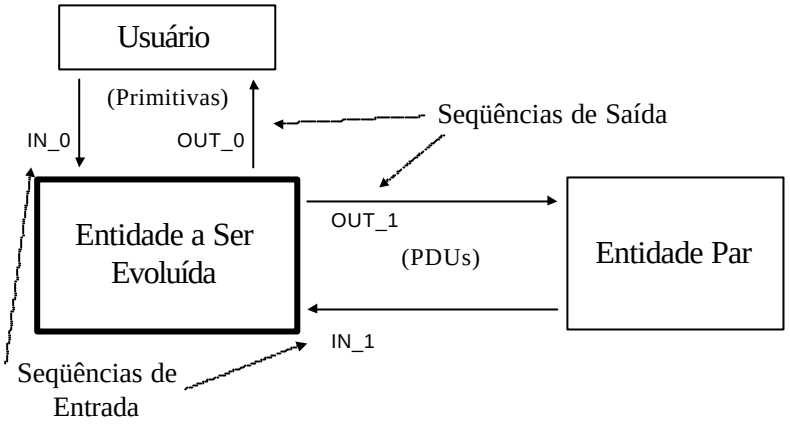


Figura 5.5: Modelo para o cálculo de aptidão na metodologia de síntese de protocolos de comunicação.

5.3.5 Derivação das Seqüências de Treinamento

Uma *seqüência de treinamento ST* é uma seqüência de pares de eventos de entrada e saída *corretos* $< v_1/v_1', v_2/v_2', \ldots, v_L/v_L' >$, onde $v \in V$, $v' \in V'$ e L é o comprimento da seqüência. Para a derivação completa de uma *ST*, os seguintes *elementos de protocolo* devem ser definidos [91]:

(1) Especificação do serviço do protocolo;
(2) Restrições do protocolo; e
(3) PDUs e suas respectivas associações às primitivas de serviço.

O protocolo orientado à conexão, introduzido na subseção 5.3.2, é utilizado como exemplo. Nas asserções seguintes, ES_G denota a especificação global do serviço do protocolo orientado à conexão, EP_E denota a entidade emissora do protocolo orientado à conexão, ES_E denota a especificação do serviço da EP_E e $EPRO_E$ denota a especificação de protocolo da EP_E. Da mesma forma, EP_R denota a entidade receptora do protocolo orientado à conexão, ES_R denota a especificação do serviço da EP_R e $EPRO_R$ denota a especificação de protocolo da EP_R.

Os elementos de protocolo do protocolo orientado à conexão são detalhados na se-

qüência: **(1)** a ES_G, assumida seqüencial, isto é, há uma relação de causalidade entre dois eventos consecutivos, e livre de *deadlock* e *livelock*, é mostrada na figura 5.2a; **(2)** As restrições do protocolo são: **(2.1)** cada estado do protocolo deve ser capaz de responder a todos os eventos de entrada, ou seja, o resultado do processo de síntese é uma MEF completamente especificada; **(2.2)** somente as fases de estabelecimento, reinicialização e liberação de conexão são consideradas; **(2.3)** somente a EP_E pode iniciar e reiniciar a conexão e; **(2.4)** o meio de comunicação é confiável; **(3)** As PDUs, juntamente com as primitivas de serviço empregadas na ES_E (figura 5.2b), são listadas na tabela 5.1 que define os alfabetos de entrada, $V = \{0, 1, 2, 3, 4, 5\}$, e de saída, $V' = \{0, 1, 2, 3, 4, 5, 6\}$, da EP_E. Da mesma forma, as PDUs e as primitivas de serviço empregadas na ES_R (figura 5.2c) são listadas na tabela 5.2 que define seus alfabetos de entrada e de saída; **(3.1)** A associação entre primitivas de serviço e PDUs é definida pelo conjunto {*C_Req*$_E$*-con_req, C_Resp*$_R$*-con_acc, D_Req*$_E$*-dis_ind, D_Req*$_R$*-dis_ind, R_Req*$_E$*-rei_req, R_Resp*$_R$*-rei_acc, con_acc-C_Conf*$_E$*, dis_ind-D_Ind*$_E$*, rei_acc-R_Conf*$_E$*, con_req-C_Ind*$_R$*, dis_ind-D_Ind*$_R$*, rei_req-R_Ind*$_R$}. Por exemplo, uma PDU *con_req* é enviada da EP_E para a EP_R após a ocorrência da primitiva de serviço *C_Req*$_E$, caso a EP_E se encontre no estado de repouso.

Tabela 5.1: PDUs e primitivas de serviço da EP_E.

Entrada (*V*)	**descrição**	**tipo**	**codificação**
`C_Req`	Connect_Request	Primitiva	0
`D_Req`	Disconnect_Request	Primitiva	1
`con_acc`	connect_accept	PDU	2
`dis_ind`	disconnect_indication [a]	PDU	3
`R_Req`	Reinitialization_Request	Primitiva	4
`rei_acc`	reinitialization_accept	PDU	5
Saída (*V'*)	**descrição**	**tipo**	**codificação**
`C_Conf`	Connect_Confirm	Primitiva	0
`D_Ind`	Disconnect_Indication	Primitiva	1
`con_req`	connect_request	PDU	2
`dis_ind`	disconnect_indication	PDU	3
`R_Conf`	Reinitialization_Confirm	Primitiva	4
`rei_req`	reinitialization_request	PDU	5
Null	"No output"	-	6

[a] utilizada como negação à PDU `con_req`

Tabela 5.2: PDUs e primitivas de serviço da EP_R.

Entrada (*V*)	**descrição**	**tipo**	**codificação**
C_Resp	Connect_Response	Primitiva	0
D_Req	Disconnect_Request	Primitiva	1
con_req	connect_request	PDU	2
dis_ind	disconnect_indication	PDU	3
R_Resp	Reinitialization_Response	Primitiva	4
rei_req	reinitialization_request	PDU	5
Saída (*V'*)	**descrição**	**tipo**	**codificação**
C_Ind	Connect_Indication	Primitiva	0
D_Ind	Disconnect_Indication	Primitiva	1
con_acc	connect_accept	PDU	2
dis_ind	disconnect_indication	PDU	3
R_Ind	Reinitialization_Ind	Primitiva	4
rei_acc	reinitialization_accept	PDU	5
Null	"No output"	-	6

Heurística de Derivação das Seqüências de Treinamento

A heurística de derivação de seqüências de treinamento é fundamentada em técnicas de geração de testes de conformidade de protocolos.

Em um procedimento de teste, executa-se um sistema (programa) com a intenção de encontrar erros. Uma estratégia ótima de teste, capaz de detectar todos os possíveis erros, é o teste exaustivo, embora seja, na maioria das vezes, inexeqüível. Então, as técnicas de teste definem estratégias com o objetivo de encontrar um subconjunto de casos de teste com alta probabilidade de detecção de erros. Identificam-se, na literatura, diversos métodos formais de geração de seqüências de teste, derivadas diretamente de uma especificação do sistema através de MEFs. Dentre eles, destacam-se [108]: (a) *Transition Tour* (método-T); (b) *Distinguishing Sequence* (método-D); (c) *Characterizing Sequence* (método-W, onde *W* é o conjunto de caracterização); e (d) *Unique Input/Output (UIO) Sequence* (método-UIO).

O método-T é o mais simples e de mais fácil implementação. Sua idéia básica é fazer uma excursão persuasiva pelas transições da MEF. O método-T gera uma seqüência

de entradas que inicia em um dado estado s_i, percorre, pelo menos por uma vez, todas as transições da MEF e retorna a s_i. Este método é capaz de detectar erros de saída, embora não seja capaz de detectar erros de transferência de estados. Isto porque o estado alcançado após a execução de uma transição não é checado. Os demais métodos acima citados são capazes de suprir esta deficiência do método-T, embora apenas possam ser aplicados a MEFs com propriedades particulares e com o custo de uma alta complexidade de implementação (se comparados ao método-T).

A heurística de derivação de seqüências de treinamento proposta neste trabalho baseia-se no método-T. A deficiência acima mencionada é compensada pela realização do teste de equivalência entre autômatos no final da evolução (Cf. figura 5.4). Uma das formas de criar seqüências de teste no método-T é gerar aleatoriamente eventos de entrada até que todas as transições da máquina de estados tenham sido visitadas [109]. No método proposto neste trabalho percorre-se a MEF da especificação do serviço do protocolo, a partir do estado inicial, com este mesmo objetivo. Quando houver mais de uma opção de transição, esta será decidida aleatoriamente. Por exemplo, após a seleção da primitiva $!C\_Conf_E$ (vide figura 5.2a), três primitivas podem ocorrer (D_Req_E, D_Req_R ou R_Req_E). A primitiva será escolhida aleatoriamente. O procedimento prossegue até que todas as transições da MEF estejam presentes na seqüência.

A seguir é apresentada uma heurística para gerar seqüências de treinamento (*STs*) a partir do modelo global do serviço do protocolo. Após cada passo é mostrado um exemplo de formação de uma *ST* do lado emissor do protocolo orientado à conexão, descrito na subseção 5.3.2 e na presente subseção.

Passo_1: Uma seqüência de primitivas de serviço é criada a partir da ES_G (figura 5.2a). Nesse passo é desejável que todos os caminhos da ES_G estejam presentes. Utiliza-se, para este propósito, o método-T acima discutido. Note que, alternativamente, pode-se utilizar um gerador automático de seqüências de teste programado para gerar seqüências corretas a partir da ES_G.

Exemplo: Percorrendo os estados 1, 2, 3, 4, 5, 7, 1, 2, 3, 7, 1, 2, 3, 4, 5, 8, 9, 10, 5 e 6 da ES_G (figura 5.2a), obtém-se a seguinte seqüência de primitivas de serviço:

<C_Req_E, C_Ind_R, C_Resp_R, C_Conf_E, D_Req_R, D_Ind_E, C_Req_E, C_Ind_R, D_Req_R, D_Ind_E, C_Req_E, C_Ind_R, C_Resp_R, C_Conf_E, R_Req_E, R_Ind_R, R_Resp_R, R_Conf_E, D_Req_E, D_Ind_R>.

Passo_2: A PDU associada é inserida entre os pares de primitivas de serviço consecutivas que ocorrem em *sites* diferentes, exceto quando uma escolha entre primitivas de serviço descendentes é realizada pelo usuário.

Exemplo: A partir da associação primitiva/PDU da definição **3.1**, as PDUs, destacadas em negrito, são inseridas na seqüência, conforme abaixo:

<C_Req_E, **con_req**, C_Ind_R, C_Resp_R, **con_acc**, C_Conf_E, D_Req_R, **dis_ind**, D_Ind_E, C_Req_E, **con_req**, C_Ind_R, D_Req_R, **dis_ind**, D_Ind_E C_Req_E, **con_req**, C_Ind_R, C_Resp_R, **con_acc**, C_Conf_E, R_Req_E, **rei_req**, R_Ind_R, R_Resp_R, **rei_acc**, R_Conf_E, D_Req_E, **dis_ind**, D_Ind_R>.

Passo_3: Uma nova seqüência de pares de eventos de entrada e saída é gerada após a projeção da seqüência acima obtida (Passo 2) no alfabeto de entrada *V* da EP_E.

Exemplo: Considerando o alfabeto de entrada definido na tabela 5.1, ou seja, *V* = {*C_Req, D_Req, con_acc, dis_ind, R_Req, rei_acc*}, obtém-se a seguinte seqüência:

<C_Req_E/con_req, con_acc/C_Conf_E, dis_ind/D_Ind_E, C_Req_E/con_req, dis_ind/D_Ind_E, C_Req_E/con_req, con_acc/C_Conf_E, R_Req_E/rei_req, rei_acc/R_Conf_E, D_Req_E/dis_ind>.

Passo_4: É contemplada a restrição **2.1** pela inclusão, na *ST*, de eventos de entrada escolhidos aleatoriamente a partir do conjunto de eventos de entrada não especificados (ou não esperados), V_{NE}. Este conjunto é deduzido por $V_{NE} = V - V_E$, onde V_E é o conjunto de eventos de entrada esperados obtido a partir da figura 5.2 e da definição 3.1. Tais eventos devem ser inseridos em posição aleatória na seqüência de modo a exercitar o processo evolutivo. Nesse caso, a EP_E responde com saída nula (*Null*). Uma exceção é feita em relação aos eventos de entrada *dis_ind* e D_Req_E. Nesse caso, a EP_E deve responder com D_Ind_E e *dis_ind*, respectivamente, e o evento de en-

trada subseqüente deve ser a primitiva de serviço C_Req_E, a fim de satisfazer à ES_G. Note, nesse caso, que o resto da seqüência deve ser reescrito.

Exemplo: Os eventos inseridos nesse momento estão destacados em negrito.

<C_Req_E/con_req, **C_Req_E/Null**, con_acc/C_Conf_E, **rei_acc/Null**, dis_ind/D_Ind_E, **con_acc/Null**, C_Req_E/con_req, dis_ind/D_Ind_E, **R_Req_E/Null**, C_Req_E/con_req, **dis_ind/D_Ind_E**, C_Req_E/con_req, **C_Req_E/Null**, con_acc/C_Conf_E, **con_acc/Null**, **D_Req_E/dis_ind**>.

A *ST* obtida pelos procedimentos acima descritos é apresentada abaixo utilizando a codificação da tabela 5.1.

ST = <0/2, 0/6, 2/0, 5/6, 3/1, 2/6, 0/2, 3/1, 4/6, 0/2, 3/1, 0/2, 0/6, 2/0, 2/6, 1/3>.

5.3.6 Estudo de Caso: Protocolo Orientado à Conexão

5.3.6.1 Descrição do Projeto

O objetivo deste experimento é sintetizar o protocolo orientado à conexão cujo modelo de serviço foi apresentado na subseção 5.3.2 e cujas restrições foram descritas na subseção 5.3.5. Ao final de sua execução, o sistema deve produzir especificações de protocolo das entidades emissora e receptora corretas, isto é, equivalentes ao modelo de serviço do protocolo fornecido na entrada.

5.3.6.2 Definição da BNF

A seguinte BNF foi definida para resolver o problema em questão:

```
S           ::=   <fe>;
<fe>        ::=   "S" <expr> "E";
<expr>      ::=   <if_stat>|<expr> <if_stat>;
<if_stat>   ::=   <next_st><out> <next_st><out> <next_st><out>
                  <next_st><out> <next_st><out> <next_st><out>;
```

```
<next_st>   ::=   "0" | "1" | "2" | "3" | "4" | "5";
<out>       ::=   "0" | "1" | "2" | "3" | "4" | "5" | "6";
```

A BNF acima descrita estabelece 6 eventos de entrada (quantidade fixa de pares de símbolos não terminais <*next_st*><*out*>), 7 possíveis eventos de saída (símbolos terminais "0" a "6") e número máximo de 6 estados (símbolos terminais "0" a "5").

5.3.6.3 Principais Atributos de PG

As principais características para este sistema de PG estão definidas na tabela 5.3.

Tabela 5.3: Características de PG para o experimento de síntese de protocolos.

Objetivo:	Criar uma MEF, com número mínimo de estados, que descreva corretamente a entidade de protocolo (EP_E ou EP_R) a partir de um conjunto de *STs*.
Símbolos terminais:	Saídas: 0, 1, 2, 3, 4, 5 e 6; identificadores de estado: 0, 1, 2, 3, 4 e 5; delimitadores de *string*: "S" (*Start*) e "E" (*End*).
Símbolos não terminais:	<*fe*>, <*expr*>, <*if_stat*>, <*next_st*> e <*out*>.
Seqüências de treinamento (*STs*):	Dezoito *STs* de 32 bits, derivadas da ES_G do protocolo orientado à conexão e de um conjunto de PDUs associadas (Cf. subseção 5.3.5).
Aptidão:	Número de acertos de saída das dezoito *STs*, ponderado pelo número de estados utilizados para implementar a MEF de acordo com a equação 4.6 (w_i = 1).
Método de seleção:	Proporcional à aptidão.
Parâmetros principais (na condição GR > 0,1):	M=500, G=3000, p_c=0,65 (cruzamento em dois pontos), p_m=0,05, p_r=0,3, com elitismo.

5.3.6.4 Resultados

O experimento foi realizado com uma população de 500 indivíduos que evoluiu através de 3000 gerações. O comprimento das seqüências de treinamento (*STs*) foi calculado utilizando-se 6 eventos de entrada (Cf. tabelas 5.1 e 5.2) e número de estados (*S*) estimado em 5, o que resultou em *STs* com 168 pares de eventos de entrada e saída (Cf.

equação 4.4). Sabe-se, por experimentos, que N = 3 seqüências de 168 bits (que corresponde, aproximadamente, a N = 18 seqüências de 32 bits) já são suficientes para gerar a especificação correta do protocolo em questão.

No entanto, este experimento utilizou, inicialmente, N = 14 seqüências de 32 bits com o objetivo de testar a metodologia em todo o seu fluxograma (Cf. figura 5.4). Desta forma, N poderá vir a ser aumentado em função do teste de equivalência (Cf. subseção 5.3.4). As *STs* foram geradas a partir da especificação do serviço do protocolo da figura 5.2, utilizando o método de derivação apresentado na subseção 5.3.5.

Na geração 869 de uma execução particular, o sistema produziu indivíduos que atenderam totalmente às restrições impostas pelas *STs* de ambos os sistemas evolutivos, lado emissor e lado receptor, com total de 14 × 32 = 448 acertos para cada um deles. Neste momento, foi realizado o teste de equivalência através da ferramenta CWB. Para este propósito, as MEFs com as especificações de protocolo da EP_E e da EP_R, assim como a MEF da ES_G, fornecida na entrada do sistema, foram traduzidas manualmente para CCS. Neste caso, a ferramenta CWB não verificou a equivalência entre os autômatos. A figura 5.6 mostra a MEF que especificou incorretamente o lado emissor do protocolo orientado à conexão. Esta especificação apresenta um estado redundante (o estado 1), o que não invalida o protocolo. Note, no entanto, que a transição do estado 0 para o estado 3 (em negrito), conseqüência da entrada 4 (PDU *rei_req*), gera erro semântico.

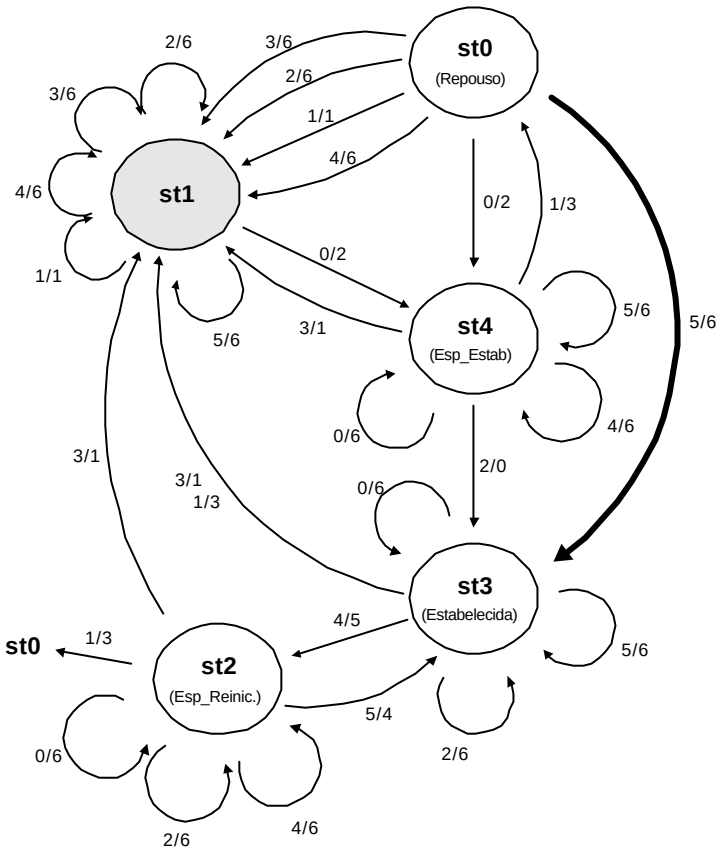
Como conseqüência, as seqüências de treinamento de ambos os sistemas evolutivos foram incrementadas e se repetiu todo o procedimento. Finalmente, a condição N = 17 (com total de 17 × 32 = 544 acertos) verificou a equivalência entre os autômatos gerando corretamente a especificação do protocolo. A figura 5.7 mostra a máquina Mealy resultante que descreve a especificação correta da entidade emissora do protocolo orientado à conexão ($EPRO_E$) utilizando um grafo de transição de estado (STG). A figura 5.8 mostra a máquina Mealy resultante que descreve a especificação correta da entidade receptora do protocolo orientado à conexão ($EPRO_R$). Estes indivíduos foram sintetizados com apenas 4 estados.

A figura 5.9 mostra a curva de aptidão do melhor indivíduo do lado emissor do protocolo. Observa-se, nesta curva, uma evolução "comportada" até a geração 869, momento em que é realizado o primeiro teste de equivalência (frustrado) e se inicia o in-

crementó do número de seqüências de treinamento. O sistema operou com $N = 15$, $N = 16$ e $N = 17$ para enfim fornecer a especificação correta do protocolo. Neste período da evolução ($G = 869$ e $G = 925$), nota-se três saltos do valor da aptidão do melhor indivíduo conseqüência da inclusão de novas seqüências de treinamento.

Tabela de Codificação

ENTRADA	SAÍDA
C_Req -0	C_Conf -0
D_Req -1	D_Ind -1
con_acc -2	con_req -2
dis_ind -3	dis_ind -3
R_Req -4	R_Conf -4
rei_acc -5	rei_req -5
	error_ind -6



Cromossomo:

421116161636 421116161616 261326112634 361336112536 461330114646

Figura 5.6: Solução incorreta para a EP_E, com um estado redundante (círculo sombreado).

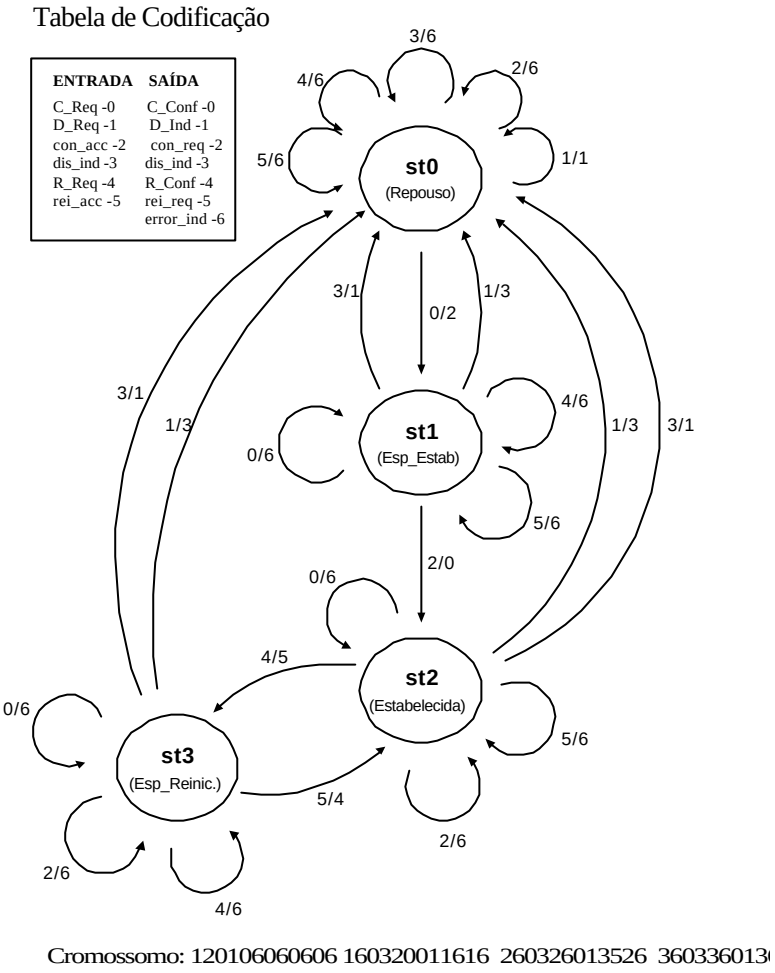


Figura 5.7: *EPRO*$_E$, utilizando STG, do melhor indivíduo.

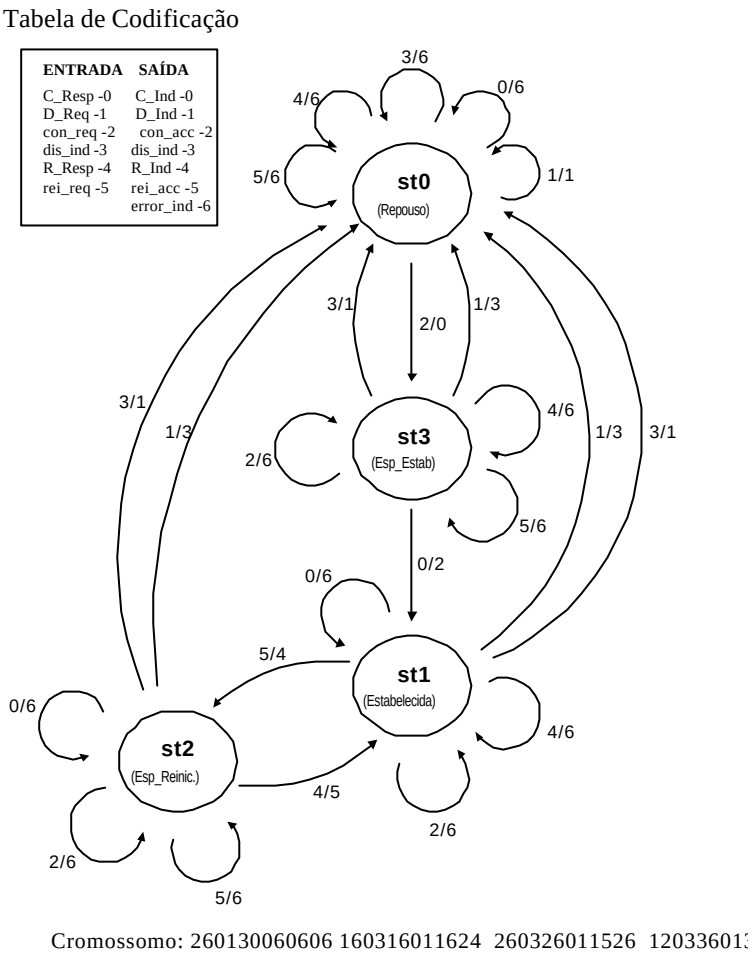


Figura 5.8: *EPRO*$_R$, utilizando STG, do melhor indivíduo.

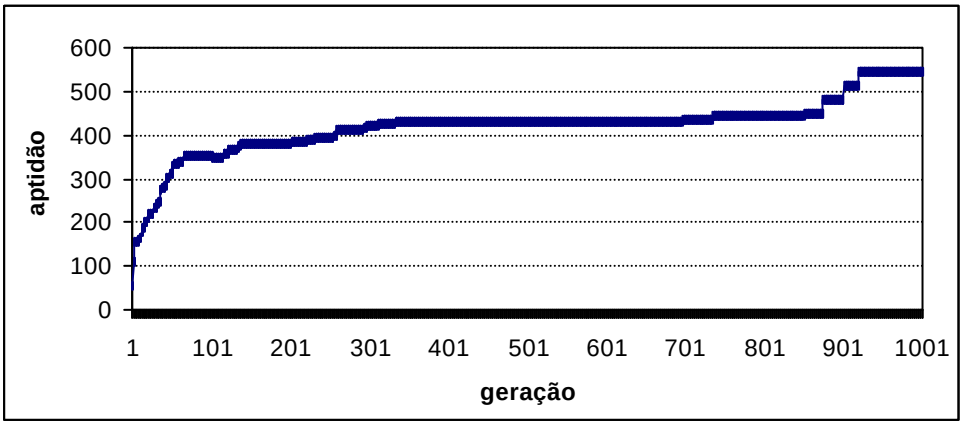


Figura 5.9: Curva de aptidão do melhor indivíduo para o experimento de síntese de protocolos de comunicação (lado emissor).

5.4 Projeto de Protocolos de Comunicação a Partir de Cenários de Interação

Nos últimos anos, várias abordagens têm sido propostas para descrever aspectos comportamentais de sistemas e serviços de telecomunicações. Freqüentemente, métodos formais são apresentados para solucionar problemas através de notações matemáticas e técnicas de verificação. Não obstante, a difusão destes métodos na indústria e em órgãos de normalização tem sido inexpressiva [110].

Por outro lado, muita atenção tem sido dispensada a métodos semiformais, ou meramente informais, com o objetivo de promover o contato com o cliente da aplicação que se comunica através de formatos de representação mais intuitivos que formais. A captura, ou extração, de requisitos de entidades diretamente envolvidas no sistema (*stakeholders*: projetistas, gerentes, consumidores, etc.) é um dos fundamentos da *engenharia de requisitos* (*requirements engineering*) [111], que também prevê uma etapa posterior de formalização desta informação direcionada à implementação.

Dentre as abordagens semiformais, destacam-se os cenários de interação. Modelos baseados em cenários são adequados para a descrição de aspectos operacionais que são críticos em sistemas reativos e sistemas de telecomunicações e podem ser introduzidos de forma interativa e incremental. Esse é um aspecto importante, já que novos serviços são continuamente disponibilizados em sistemas existentes. Além disso, cenários de interação típicos de um sistema hipotético são mais fáceis de serem obtidos que propriedades ou metas quando o projeto do sistema encontra-se na fase inicial [110].

Um *cenário de interação*, também conhecido como *diagrama de interação* ou *diagrama de traços de eventos*, é composto por traços, isto é, seqüências tipicamente finitas de ações ou eventos, que fornecem uma descrição parcial do comportamento do sistema visto pelos seus usuários e por sistemas relacionados. Através de cenários, seqüências de ações são especificadas ao invés de seqüências de estados.

Especificações orientadas a estado e descrições orientadas a interação provêem visões ortogonais de sistemas. Autômatos representam projeções do comportamento do sistema em *componentes*, enquanto cenários de interação representam projeções do

comportamento do sistema em *serviços*. Um autômato revela, em certo grau, como um comportamento particular é implementado; deste modo, sua posição no processo de desenvolvimento do sistema é mais próxima de projeto e de implementação que de captura de requisitos [112].

Claramente, a aplicação de cenários de sistemas, tipicamente empregados em sistemas de engenharia reversa, dentro de uma abordagem de engenharia direta requer uma transformação precisa entre modelos. Para este fim, várias estratégias de construção objetivando a criação de modelos orientados à implementação a partir de cenários de interação têm sido propostas (vide próxima subseção). No entanto, esses métodos são, na sua maioria, analíticos e, freqüentemente, não tratam o problema da *sobregeneralização* do modelo sintetizado.

A metodologia apresentada na seção anterior abordou a síntese de protocolos, isto é, a geração de especificações de protocolo a partir de especificações de serviço do protocolo. Ocorre que, na maioria das vezes, a especificação formal do serviço do protocolo não está disponível. Comumente, o processo de construção de novos protocolos concentra-se nos serviços a serem fornecidos pelo sistema ao usuário, revelados na forma de casos de uso, relegando os agentes constituintes do sistema. A presente metodologia gera protocolos de comunicação a partir de especificações feitas através de descrições semiformais. Técnicas evolutivas são utilizadas para gerar máquinas de estados finitos mínimas, determinísticas e completamente especificadas utilizando diagramas visuais, isto é, cenários, como entrada do sistema.

5.4.1 Trabalhos Relacionados

Várias estratégias para traduzir MSCs (*Message Sequence Charts*) em autômatos têm sido propostas na literatura recente [112, 113]. O sistema SCED (*SCenario EDitor*) [113] pressupõe que a capacidade de enviar uma mensagem específica identifica univocamente o estado de um componente. O algoritmo é executado em vários estágios, processando traços de eventos resultantes da concatenação de todos os cenários da especificação. A complexidade deste algoritmo é, no pior dos casos, exponencial em relação ao

comprimento dos MSCs de entrada devido ao potencial *backtracking* utilizado para obter máquinas de estados finitos mínimas e determinísticas. SCED pode gerar autômatos com comportamento mais geral que os próprios cenários de entrada como conseqüência da fusão de cenários em caminhos de transição/estado (sobreposição de cenários). Nestes casos de sobregeneralização, cenários "implicados" (vide [114] para definição e exemplo) podem surgir como resultado de interações não esperadas entre máquinas de estados sintetizadas. De fato, enquanto exemplos num típico sistema de inferência indutiva podem ser ambos positivos ou negativos, os cenários de interação, que representam tão somente dados positivos, fornecem informação insuficiente para inferência [76, 115].

Krüger [112] derivou autômatos a partir de um conjunto de MSCs pela aplicação sucessiva de transformações realizadas em quatro etapas: 1ª) *Projeção* dos MSCs no componente; 2ª) *Normalização* para a determinação dos seguimentos dos caminhos de transição definidos pelos MSCs projetados; 3ª) *Transformação* em autômatos através da conversão de cada mensagem em uma transição e, conjuntamente, da adição de estados intermediários, e 4ª) *Otimização* do autômato resultante. Com exceção do procedimento de otimização, estas transformações são lineares em relação ao comprimento dos MSCs de entrada. Uma característica importante desta abordagem é que o projetista pode guiar a qualidade da saída do algoritmo de inferência fornecendo conhecimento de projeto na forma de *guardas* (informações de estado). Ao contrário do SCED, entretanto, o autômato produzido é não determinístico e não mínimo por construção, já que o sistema considera a otimização como uma etapa distinta.

Alur *et al*. [114] apresentaram um método para inferir cenários implicados por um conjunto de MSCs não explicitamente descritos. Em alguns casos, os cenários implicados podem ser inconsistentes com o propósito do projetista. Tais cenários, originalmente escondidos, proporcionam informação útil ao projetista, vez que interações indicativas de comportamento indesejado podem ser reveladas. Este trabalho foi estendido em [116] que utilizou cenários de interação mais expressivos, descritos através de hMSCs (*high-level MSCs*).

5.4.2 *Message Sequence Charts* (MSCs) e suas Interpretações

MSCs (Diagramas de Seqüências de Mensagens), e suas extensões, são notações de cenários comumente utilizadas para a captura de requisitos no domínio dos sistemas de telecomunicações [110, 112]. Os MSCs enfatizam o aspecto de coordenação entre componentes de sistemas de forma bastante intuitiva. Um MSC é um cenário gráfico constituído de *eixos verticais* que representam instâncias reativas (entidades de comunicação) e de *setas*, direcionadas da instância emissora para a instância receptora, denotando um evento de comunicação. Essas setas recebem *rótulos* que identificam a mensagem trocada entre as instâncias. A figura 5.10a apresenta um cenário de comunicação que utiliza um dialeto básico de MSCs. Nesta figura, uma seqüência de interação entre três entidades (*usuário (Usu), emissor (Emi)* e *receptor (Rec)*) especifica um protocolo orientado à conexão simplificado. A caixa rotulada *Ativado* representa um *símbolo de condição* que é utilizado para indicar que uma ou mais instâncias atenderam a uma determinada condição antes ou depois de fazerem parte de uma seqüência de interação.

Já que os MSCs descrevem, usualmente, comportamentos parciais de sistemas, levanta-se a questão sobre quão completa é a informação contida numa especificação com MSCs e como transitar entre comportamentos permitidos e mandatários. Então, faz-se indispensável entender corretamente as *interpretações* dos MSCs em relação ao sistema que está sendo projetado. Em [112] definem-se quatro possíveis interpretações: 1ª) *Existencial* - o comportamento pode mas não necessita ocorrer durante a execução do sistema; 2ª) *Universal* - o comportamento deve ocorrer em todas as execuções; 3ª) *Exata* - proíbe explicitamente outros comportamentos além dos especificados através dos MSCs e 4ª) *Negação* (anti-cenários) - utilizados para descrever comportamentos indesejáveis ou proibidos. A interpretação exata vai além das interpretações existencial e universal no sentido de que ela não apenas estabelece o que *pode* ou *deve* ocorrer (o comportamento explicitamente representado pelos MSCs), mas também esclarece o que *não deve* ocorrer (todo o restante). Nesse sentido, a interpretação exata assegura implicitamente a ocorrência de MSCs com contra-exemplos, como desejável em abordagens de inferência indutiva (vide [115]). Conseqüentemente, esta interpretação pode ser utilizada para inibir a geração de autômatos com comportamento sobregeneralizado.

A figura 5.10b mostra o comportamento da entidade emissora do protocolo orientado à conexão através de uma MEF Mealy, correspondente à interpretação exata do MSC mostrado na figura 5.10a. Considera-se, neste caso, que a ocorrência de eventos que não os especificados está proibida. Nesta figura, o envio, para o meio de comunicação ou para o usuário, e a recepção, do meio de comunicação ou do usuário, de um evento *e* são denotados !*e* e ?*e*, respectivamente.

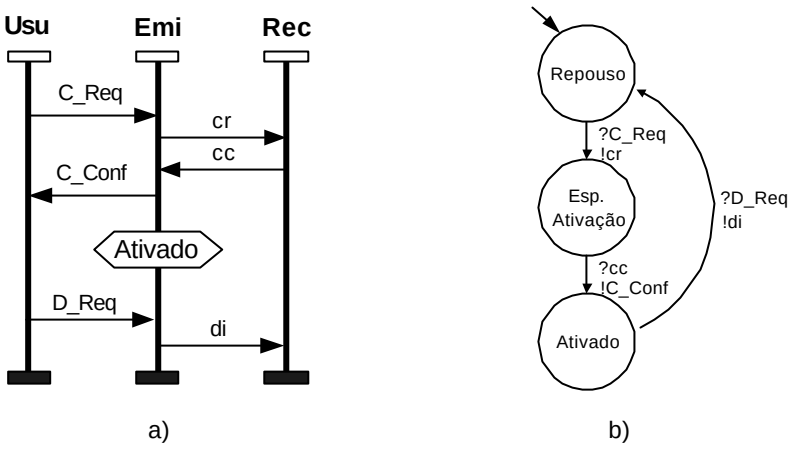


Figura 5.10: Modelos de especificação: (a) MSC; e (b) MEF (emissor).

5.4.3 Metodologia

A metodologia de projeto de protocolos de comunicação a partir de cenários de interação [13] trata o problema de gerar uma MEF determinística, possivelmente mínima em relação ao número de estados, consistente com uma determinada amostra de comportamento do sistema na forma de seqüências de entrada e saída extraídas de MSCs. As seqüências de treinamento são derivadas através da projeção da especificação em MSCs na entidade de interesse. A metodologia é mostrada na figura 5.11. Seu fluxograma é semelhante àquele apresentado na seção 4.4 (figura 4.3), apenas se diferindo no que tange aos blocos sombreados. A presente metodologia inclui uma heurística para gerar seqüências parciais de entrada e saída, ou seqüências de treinamento, a partir de um conjunto de MSCs, a ser apresentada na subseção seguinte.

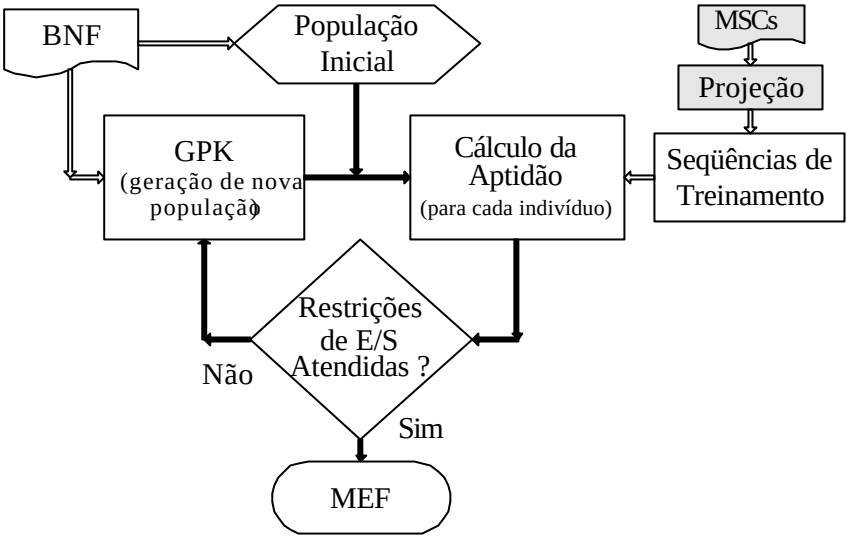


Figura 5.11: Metodologia de projeto de protocolos de comunicação a partir de cenários de interação.

5.4.4 Derivação das Seqüências de Treinamento

Um conjunto de seqüências de treinamento (*STs*) será utilizado como meta de evolução de um autômato que deverá reproduzir o comportamento da entidade de protocolo na forma de traços. O método de derivação de *STs* é baseado na interpretação exata de cenários de interação utilizando MSCs (vide subseção 5.4.2), interpretação esta também adotada em outras abordagens de construção a partir de cenários de interação [112, 113]. O processo de derivação de *STs* a partir de uma coleção de MSCs é realizado em duas etapas:

1ª Etapa: Geração de *STs* com informação positiva, através da projeção de todos os MSCs sobre o objeto para o qual a máquina de estados finitos será sintetizada. A projeção é realizada percorrendo a linha vertical do objeto do topo à base: cada evento recebido constitui um evento de entrada (v_i) e cada evento enviado constitui um evento de saída (v_i'). Na seqüência, os eventos de entrada e saída, ordenados seqüencialmente, são agrupados em pares de entrada e saída, constituindo uma *ST*. A composição seqüencial de traços originados de dois ou mais MSCs contendo o obje-

to de interesse é realizada por meio de símbolos de condição. A combinação randômica de traços segundo esta estratégia permite a geração de *STs* diversificadas e alongadas.

A figura 5.12 mostra uma especificação por meio de MSCs do protocolo orientado à conexão. Nesta figura, a barra na notação *evento-A/evento-B* indica que um dos dois eventos, *A* ou *B*, pode ocorrer. Estes MSCs possuem dois símbolos de condição, *Ativado* e *Repouso*, que são utilizados para acoplar os MSCs. As seguintes *STs* foram geradas a partir dos procedimentos descritos na primeira etapa, pela seleção da entidade emissora como objeto alvo na especificação apresentada na figura 5.12:

ST_1: <C_Req/cr, cc/C_Conf, D_Req/dr> (MSC1 e MSC5);
ST_2: <C_Req/cr, cc/C_Conf, R_Req/rr, rc/R_Conf> (MSC1 e MSC3);
ST_3: <C_Req/cr, dr/D_Ind, C_Req/cr, cc/C_Conf, dr/D_Ind> (MSC2, MSC1 e MSC5).

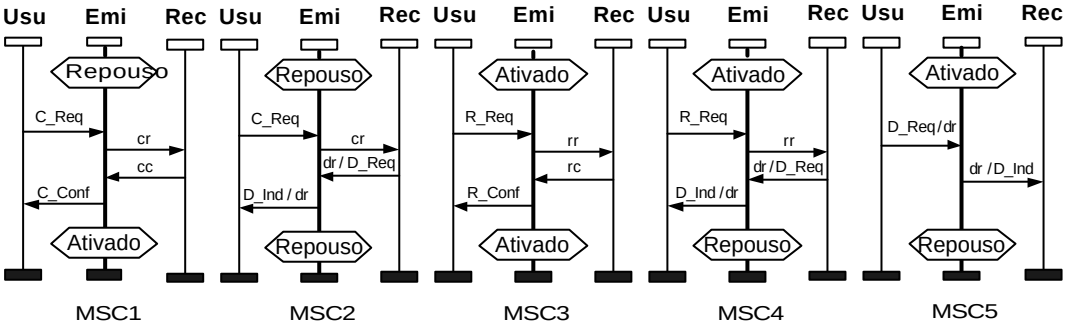


Figura 5.12: Especificação do protocolo orientado à conexão utilizando MSCs.

É importante notar que as *STs* produzidas na primeira etapa não incluem eventos de entrada não especificados (ou não esperados), resultando em informação incompleta para a correta inferência de máquinas de estados. O atendimento à propriedade de completude faz-se aqui necessário para assegurar a equivalência entre a MEF sintetizada e a MEF desconhecida, da qual as amostras de traços foram geradas [117]. A completude implica que haja uma transição para todo evento de entrada. A construção de traços com esta suposição garante que eles irão fornecer informação suficiente sobre a máquina de estados para o processo de inferência.

A conversão de máquinas de estados parcialmente especificadas em máquinas de estados completamente especificadas pode ser realizada, basicamente, de duas maneiras [118]: (a) fazendo com que a máquina de estados permaneça no estado atual sem produzir saída (ou gerando saída nula) para todo evento de entrada não especificado; ou (b) forçando a transição para um "estado de erro". A implementação da primeira opção em traços é direta, como mostra a etapa seguinte.

2ª Etapa: Inserção de informação negativa nas *STs* obtidas na primeira etapa através da adição aleatória de eventos de entrada não especificados entre pares de eventos de entrada e saída. Nesse caso, a saída nula completa o par com a entrada adicionada, forçando o autômato a executar um laço. O conjunto de eventos de entrada não especificados (V_{NE}) é deduzido a partir da interpretação exata dos MSCs: $V_{NE} = V - V_E$, onde V_E é o conjunto de eventos de entrada esperados obtido da especificação em MSCs. Supõe-se que os eventos de entrada não especificados ocorram segundo uma distribuição de probabilidade uniforme, isto é, $P(E_i) = 1/N$ para $E_i \in V_{NE}$ e $i = 1,\ldots,N$.

Esta etapa garante que todo estado é capaz de tratar todas as entradas, isto é, o processo de construção induz a uma MEF completamente especificada. Este tipo de *relaxamento* da interpretação exata dos MSC, ou seja, a adição de comportamento não explicitamente especificado por meio de MSCs, foi discutido em [112] e tem o objetivo, neste trabalho, de evitar a sobregeneralização do comportamento da entidade a ser inferida. As seguintes *STs* foram obtidas ao final da segunda etapa:

ST_1:<C_Req/cr, **rc/Null**, cc/C_Conf, **cc/Null**, **rc/Null**, D_Req/dr>;
ST_2:<C_Req/cr, **C_Req/Null**, cc/C_Conf, **cc/Null**, R_Req/rr, **D_Req/Null**, rc/R_Conf>;
ST_3:<**R_Req/Null**, C_Req/cr, **rc/Null**, **R_Req/Null**, dr/D_Ind, **rc/Null**, **cc/Null**, C_Req/cr, **R_Req/Null**, cc/C_Conf, **cc/Null**, dr/D_Ind>.

O método de derivação acima proposto não permite a associação de mais de um evento de saída a um único evento de entrada. Entretanto, saídas múltiplas podem ser codificadas como um único evento de saída. Por outro lado, em caso de haver dois eventos de entrada consecutivos, uma saída nula é inserida entre eles.

5.4.5 Estudo de Caso: Protocolo Orientado à Conexão

5.4.5.1 Descrição do Projeto

Este experimento utilizou o protocolo orientado à conexão que foi discutido na seção anterior. No entanto, diferentemente daquele experimento, o objetivo do presente experimento é gerar somente a especificação da entidade de protocolo do lado emissor a partir de uma especificação inicial com MSCs.

5.4.5.2 Definição da BNF

Utiliza-se a mesma BNF definida no item 5.3.6.2.

5.4.5.3 Principais Atributos de PG

Utilizam-se os mesmos atributos de PG definidos na tabela 5.3 (item 5.3.6.3).

5.4.5.4 Resultados

O trabalho concentrou-se no problema de gerar uma MEF com número mínimo de estados, consistente com as *STs* geradas a partir da especificação em MSCs (figura 5.12) através do método de derivação apresentado na subseção 5.4.4. O comprimento e a quantidade de seqüências de treinamento (*STs*) foram estabelecidos da mesma forma que no item 5.3.6.4 (L = 32 e N = 18). Utilizou-se a função de aptidão definida pela equação 4.6. Foram realizados experimentos com uma população de 500 MEFs que evoluiu até um máximo de 3000 gerações. As taxas de cruzamento, de mutação e de reprodução foram estabelecidas em p_c = 0,65, p_m = 0,05 e p_r = 0,30, respectivamente. Utilizou-se o método de seleção *Linear Rank* com elitismo. A população foi moldada

utilizando a BNF descrita no item 5.3.6.2, permitindo a geração de MEFs com até 6 estados. O fator *K(S)* foi definido pelo conjunto de valores *K(S)* = {0,01, 0,008, 0,006, 0,004, 0,002, 0,000} para *S* = {1, 2, 3, 4, 5, 6}, respectivamente. *GR* é calculado continuamente sobre as últimas 50 gerações, com exceção para as primeiras 50 gerações e ao final da evolução, isto é, em gerações com presença de indivíduos funcionalmente corretos, quando este foi estabelecido em 1. Na condição $GR \leq 0{,}1$, p_m foi elevada a 0,15 com intuito de aumentar a diversidade da população.

A figura 5.13 apresenta a curva de aptidão da melhor MEF, seu número de estados e seu gradiente *GR*, em uma execução típica. Observa-se, nesta figura, que nas primeiras 50 gerações o número de estados da melhor MEF flutua significativamente devido à alta diversidade da população inicial. O mesmo comportamento pode ser observado de *G* = 53 a *G* = 205, quando ocorre uma disputa em iguais condições entre MEFs de diferentes tamanhos, desta vez porque *GR* assume o valor zero anulando a ponderação na função de aptidão devido ao parâmetro *S* (vide equação 4.6). Finalmente, observa-se que cada acréscimo no valor de aptidão da melhor MEF está relacionado a um aumento do número de estados seguido de uma redução do mesmo (por exemplo, de *G* = 0 a *G* = 50 e de *G* = 268 a *G* = 353). Essa oscilação no número de estados para uma seqüência de gerações com aptidão crescente repetiu-se em outras execuções.

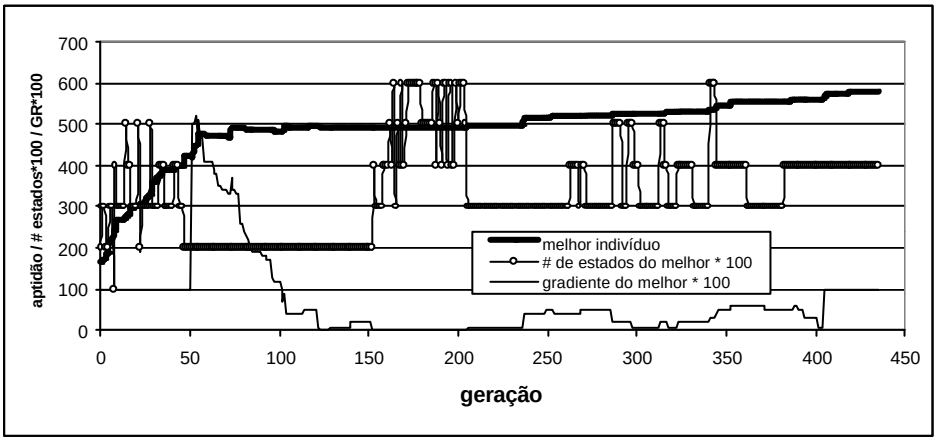


Figura 5.13: Curva de aptidão, do número de estados e de *GR* da melhor MEF.

A MEF resultante, que descreve a especificação da entidade emissora do protocolo orientado à conexão ($EPRO_E$), é mostrada utilizando um grafo de transição de estado (STG) na figura 5.14, após 576 acertos em G = 435 utilizando apenas 4 estados.

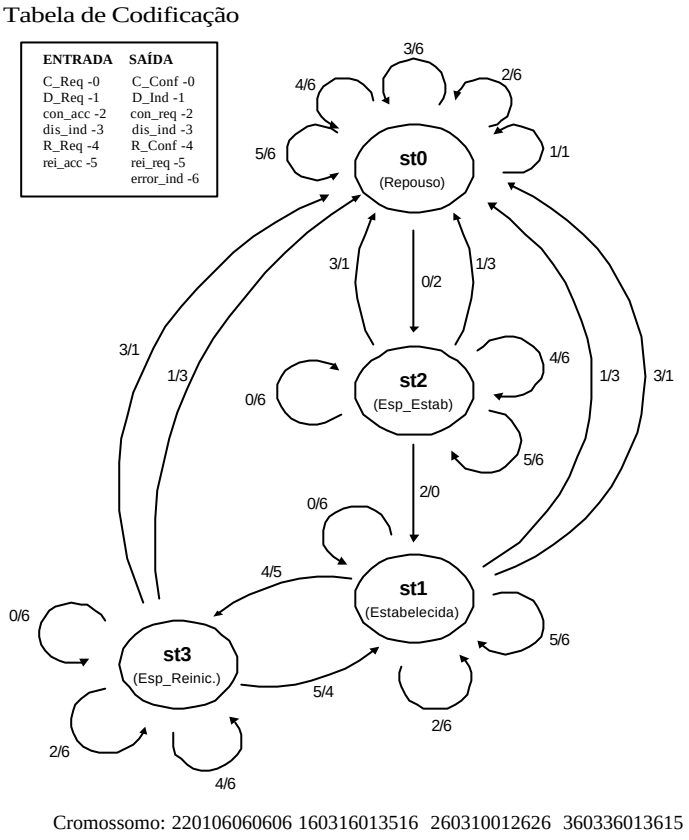


Figura 5.14: $EPRO_E$, utilizando STG, do melhor indivíduo.

Entretanto, o sistema proposto não gerou MEFs mínimas em todas as execuções de sucesso, isto é, execuções que alcançaram o número máximo de acertos para um máximo de gerações, G_{MAX}. Em alguns casos, o valor de aptidão permaneceu num ótimo b-cal durante um grande número de gerações. Nesses casos, o sistema induziu a seleção prematura de MEFs com o maior número de estados possíveis (seis) e totalizou o número de acertos com MEFs não otimizadas. Para minimizar este problema, um *procedimento de restrição de seleção* foi implementado. De acordo com este procedimento, indivíduos com dois ou mais estados em relação ao melhor indivíduo da geração anterior ficam desqualificados para o processo de seleção.

A tabela 5.4 compara os resultados de 50 execuções independentes realizadas em três diferentes configurações do sistema, quais sejam:

- Configuração 1: A função de aptidão não leva em consideração o número de estados da MEF. Usa-se, portanto, a equação 4.3;
- Configuração 2: A função de aptidão é ponderada pelo número de estados da MEF de acordo com a equação 4.6, mas exclui o procedimento de restrição de seleção;
- Configuração 3: Aplica-se a função de aptidão ponderada (equação 4.6) e o procedimento de restrição de seleção.

Esta tabela revela que: (a) sem um controle explícito do número de estados o sistema tende a gerar soluções utilizando todos os estados disponíveis; (b) a utilização de um controle explícito do número de estados, acoplado ao procedimento de restrição de seleção (configuração 3), leva o sistema a convergir para soluções ótimas, isto é, com MEFs mínimas, em 59% dos casos das execuções de sucesso, melhorando em, aproximadamente, 400% os resultados relativos à configuração que não adotou tal restrição (configuração 2); e (c) o número de execuções que não convergiram para G_{MAX} = 3000 permaneceu praticamente o mesmo para as três configurações, mostrando que o refinamento da função de aptidão em busca de MEFs mínimas não degrada a performance do sistema como um todo.

Tabela 5.4: Comparação entre três configurações de sistemas que geram protocolos de comunicação a partir de cenários de interação (50 execuções).

Configuração	Execuções que não convergiram para G_{MAX} = 3000	Execuções de sucesso (soluções funcionais) para G_{MAX} = 3000		
		4 estados	5 estados	6 estados
1. F = $\sum_{i=1}^{N} w_i H_i$	20	0	3	27
2. Equação 4.6	18	4	15	13
3. Equação 4.6 e restrição de seleção	21	17	9	3

5.5 Exemplo de Aplicação da Metodologia de Tratamento de Convergência Prematura

Este experimento tem a finalidade de demonstrar a performance da metodologia de tratamento de convergência prematura, apresentada na seção 4.6, na evolução de um sistema seqüencial de razoável complexidade. Para tanto, repetiu-se o problema apresentado na subseção anterior, aproveitando-se as seqüências de treinamento nele utilizadas. No entanto, o tamanho da população foi reduzido em relação ao utilizado naquele experimento, com intuito de fornecer condições menos favoráveis de evolução e, então, salientar a eficácia da técnica.

Foram realizados experimentos com uma população de 200 MEFs que evoluiu até um máximo de 5000 gerações. As taxas de cruzamento, de mutação e de reprodução foram estabelecidas em p_c = 0,65, p_m = 0,05 e p_r = 0,30, respectivamente. Utilizou-se o método de seleção *Linear Rank* com elitismo. A função de aptidão definida pela equação 4.3. A população foi moldada utilizando-se uma BNF semelhante à descrita na seção 4.4, permitindo, no entanto, a produção de MEFs com número fixo de 6 estados.

Indivíduos com $Sf \geq 0{,}38$ foram definidos como variantes do melhor indivíduo (Cf. equação 4.10). O cálculo do *Sf* compreendeu exclusivamente genes "próximo estado" (vide figura 4.4). De fato, a heurística de controle da busca no espaço de soluções não considerou o fator de similaridade em relação aos genes que codificam os eventos de saída (genes "saída"), vez que eles não afetam o grafo da MEF. *GR* foi calculado continuamente sobre as últimas 50 gerações e seu limiar foi ajustado em 0,1. Durante o período de ativação da heurística, o sistema funcionou com p_c = 0,60, p_m = 0,15 e p_r = 0,25. Na geração correspondente a 95% do número máximo de acertos, entretanto, a heurística foi desabilitada, pois, nesse estágio da evolução, *GR* tende naturalmente a zero. *Pf* foi definido como 0,5 (valor de aptidão penalizado em 50%), permanecendo ativado durante as 50 gerações subseqüentes. A figura 5.15 mostra as curvas de aptidão da melhor MEF em uma execução típica para três diferentes configurações do sistema, quais sejam:

- Configuração 1: Sem heurística, isto é, sem penalidade e sem alteração nos valores dos operadores genéticos quando um ótimo local é detectado;
- Configuração 2: Alteração dos valores dos operadores genéticos quando um ótimo local é detectado (penalidade não aplicada);
- Configuração 3: Heurística completa de controle da busca no espaço de soluções, isto é, penaliza-se o valor de aptidão do melhor indivíduo e de seus variantes e alteram-se os valores dos operadores genéticos quando um ótimo local é detectado.

Observa-se, na figura 5.15, que as três curvas apresentam comportamento idêntico até G = 266 (geração em que é detectado um ótimo local). Para a configuração 3 (heurística completa), de G = 267 a G = 1028, como GR reduz abaixo de 0,1 por 5 vezes, o valor da aptidão da melhor MEF recua cinco vezes (em G = 268, G = 560, G = 760, G = 904 e G = 1028) e o sistema finalmente converge para o ótimo global (F_{MAX} = 576) em G = 1230. Para a configuração 2, o sistema somente converge para o ótimo global em G = 2944. Para a configuração 1, o sistema não escapa do ótimo local (F = 554), considerando G_{MAX} = 5000.

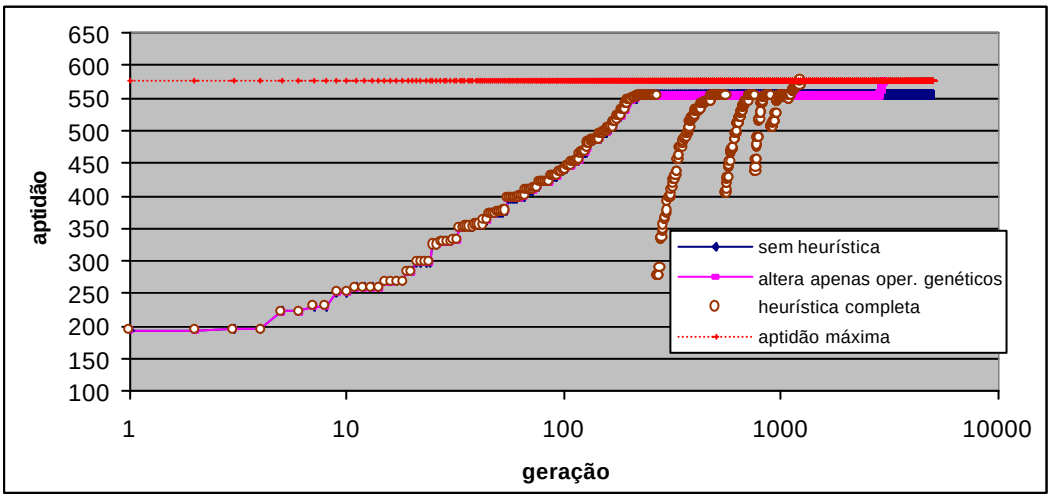


Figura 5.15: Curvas de aptidão do melhor indivíduo para três configurações de sistemas que tratam o problema da convergência prematura.

A MEF resultante da execução da configuração 3 (heurística completa), que descreve com sucesso a entidade emissora do protocolo orientado a conexão (EP_E), é mostrada na figura 5.16 através de um grafo de transição de estado (STG), após 576 acertos em G = 1230. Esta MEF contém um estado redundante (estado 4) e um estado não alcançável (estado 1). Métodos convencionais podem ser utilizados para minimização de estados.

A tabela 5.5 compara a performance entre as três configurações após 50 execuções independentes em cada configuração. Esta tabela mostra que a configuração 3 (heurística completa), que penaliza o melhor indivíduo e seus variantes quando um ótimo local é detectado, produz 38 convergências ao ótimo global, dentre 50 possíveis, para $Sf > 0{,}27$, cada uma delas evoluindo até um máximo de 1500 gerações. Esse resultado supera os resultados obtidos nas configurações 2 e 1 em 46% e 280%, respectivamente. Os resultados da tabela 5.5 indicam que a heurística de controle da busca no espaço de soluções é bastante sensível ao valor de Sf. No entanto, esta tabela também revela que todos os experimentos realizados na configuração 3, que contaram com diferentes valores de Sf, apresentaram desempenho superior aos obtidos nas demais configurações.

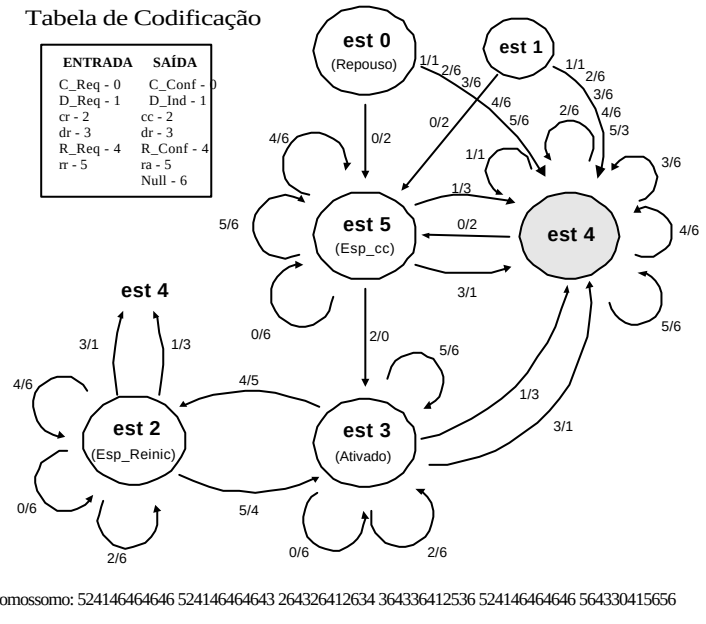


Figura 5.16: EP_E, utilizando STG, do melhor indivíduo.

Tabela 5.5: Comparação entre três configurações de sistemas que tratam o problema de convergência prematura.

	Config. 1: Sem Heurística	Configuração 2: Alteração nos valores dos oper. genéticos	Configuração 3: Heurística completa Penalidade ($Pf = 0,5$) aplicada à:			
			$Sf > 0,16$	$Sf > 0,27$	$Sf > 0,38$	$Sf > 0,55$
Número de convergências ao ótimo global para G_{MAX} =1500 (50 execuções)	10	26	32	38	32	27

5.6 Comentários

Neste capítulo foram apresentadas duas abordagens evolutivas para o projeto de protocolos de comunicação, uma sintética e a outra analítica, assim como estudos de caso referentes às metodologias propostas, apresentando e analisando os resultados experimentais.

As metodologias de síntese de protocolos de comunicação e de projeto de protocolos de comunicação a partir de cenários de interação, descritas neste capítulo, basearam-se nas metodologias de síntese de sistemas seqüenciais a partir de seqüências parciais de entrada e saída, apresentadas no capítulo 4, e em heurísticas desenvolvidas para gerar seqüências de treinamento a partir da especificação do serviço do protocolo e de cenários de interação. Tais heurísticas, embora não automatizadas, mostraram-se adequadas ao processo de inferência por incluírem nas seqüências de treinamento exemplos e contra-exemplos.

As metodologias de projeto de protocolos de comunicação desenvolvidas neste capítulo garantem de antemão a completude da especificação do protocolo, pois utilizam representação baseada em estado, gerando máquinas de estados finitos determinísticas e completamente especificadas. O atendimento a esta propriedade assegura a criação de

modelos de protocolos robustos. Na metodologia de síntese de protocolos, a correção semântica é garantida pela checagem de equivalência entre o modelo do protocolo evoluído e o modelo do serviço do protocolo, após a tradução de máquinas de estados finitos para CCS. Na metodologia de projeto de protocolos a partir de cenários de interação, a correção semântica baseia-se no fornecimento de um conjunto de seqüências de treinamento com comprimento adequado (Cf. equação 4.4), geradas com base na interpretação exata de uma especificação em MSCs fornecida na entrada.

As metodologias aqui apresentadas assumiram a restrição de comportamento estritamente seqüencial para protocolos de comunicação, que também é adotada em diversos trabalhos de pesquisa que utilizam o modelo de máquinas de estados finitos [92, 95, 96, 99]. No entanto, as modificações para se transformar uma aplicação real num sistema seqüencial devem ser realizadas no sentido de que o serviço fornecido por este sistema simplificado possa ser útil [99].

Por último, na seção 5.5, foi realizado um experimento para avaliar a performance da metodologia que trata o problema de convergência prematura, apresentada no capítulo 4 (seção 4.6). Apesar dessa abordagem não estabelecer critérios para valores iniciais de seus parâmetros, como o fator de similaridade, o fator de penalidade e o número de gerações em que o fator de penalidade deve permanecer ativado, os resultados mostraram melhora significativa na busca pelo ótimo global, a um custo computacional reduzido, para o conjunto de valores testados. Da forma análoga, a falta de critérios para a definição do parâmetro *sharing radius*, com função semelhante à do fator de similaridade, é uma restrição da técnica *fitness sharing* [88] (Cf. seção 4.6).

O próximo capítulo apresenta as conclusões do trabalho desenvolvido, destacando as vantagens e desvantagens das metodologias propostas. São apresentadas também as possíveis derivações da linha de pesquisa desenvolvida neste trabalho.

Capítulo 6

Conclusões

A computação evolutiva utiliza técnicas estocásticas de busca e otimização baseadas nos princípios da evolução de Darwin, apresentando-se como uma das principais alternativas aos métodos simbólicos de aprendizado automático [16]. Tais técnicas reduzem ou removem a dependência do conhecimento explícito no domínio do problema que se deseja resolver, favorecendo a emergência de restrições específicas de cada caso. A busca paralela por elas realizada difere das técnicas convencionais de solução de problemas pelo fato de não dependerem de decomposição e proporciona, assim, a vantagem de resolver sistemas não lineares e de natureza combinatória através de uma formulação simplificada do problema.

A programação genética, ao lado dos algoritmos genéticos, é uma das principais instâncias da computação evolutiva, caracterizando-se pela habilidade em gerar estruturas. Ela é considerada a mais eficiente técnica até agora conhecida para implementar um processo evolutivo com cromossomos de tamanho variável. As abordagens de programação genética orientada à gramática assumem grande importância, pois permitem o desenvolvimento de sistemas em uma linguagem de programação arbitrária, específica para a classe de problemas que se deseja resolver, bastando para tanto fornecer na entrada do sistema a descrição BNF de um subconjunto dessa gramática.

A preocupação fundamental deste trabalho foi investigar formas de utilização do potencial de aprendizado dos algoritmos evolutivos no desenvolvimento de sistemas digitais em geral. Em uma primeira etapa, estudou-se a síntese comportamental de sistemas digitais. A literatura relata que neste procedimento o circuito sintetizado apresenta, em geral, baixa qualidade de implementação, seja em termos de área ou de desempenho. Isto porque as ferramentas de síntese de alto nível tendem a assumir uma arquitetura alvo para facilitar as tarefas básicas de síntese, tais como alocação e escalonamento.

Com o objetivo de contornar esta deficiência, foi proposta uma metodologia de síntese de circuitos digitais combinacionais que utiliza programação genética e ferramentas de síntese de alto nível para gerar *hardware* otimizado a partir de uma descrição do sistema em alto nível de abstração. O que mais diferencia a metodologia apresentada das metodologias encontradas na bibliografia é a evolução de programas diretamente na linguagem VHDL, linguagem esta utilizada pela maioria das ferramentas de síntese hoje disponíveis. O estudo de caso apresentou os resultados esperados: a funcionalidade do sistema foi alcançada e o número de células lógicas utilizadas foi minimizado. Como desvantagem, a técnica mostrou-se lenta devido ao excessivo tempo de compilação de programas VHDL. Esse problema foi minimizado através da inserção de um pré-estágio que precedeu o processo principal de otimização (Cf. subseção 3.5.4), permitindo uma redução significativa do tempo total de execução do sistema.

Em uma segunda etapa, este trabalho abordou a síntese de sistemas seqüenciais a partir de seqüências parciais de entrada e saída através de técnicas evolutivas. Este tipo de problema encontra grande aplicação em sistemas de engenharia reversa e, em alguns casos, em processos de engenharia direta. Inicialmente, foi proposta uma metodologia básica para gerar máquinas de estados finitos a partir de eventos de interface observáveis. Em seguida, esta metodologia foi modificada para gerar máquinas com número mínimo de estados e para tratar o problema de convergência prematura. Essas metodologias foram inicialmente aplicadas na síntese de lógica seqüencial síncrona e, posteriormente, no projeto de sistemas reativos, como os protocolos de comunicação.

As abordagens de síntese de protocolos, quando é fornecida a especificação de serviço do protocolo e deseja-se obter a especificação do protocolo, estão completando mais de duas décadas. No decorrer deste tempo, os métodos de síntese de protocolos que utilizam o formalismo de máquinas de estados finitos se consolidaram. No entanto, suas execuções implicam em realizar várias transformações em autômatos até se chegar à especificação das entidades de protocolo. Este trabalho propôs um método não convencional a ser aplicado à síntese de protocolos de comunicação que reduz o número de etapas necessárias à geração das especificações das entidades de protocolo, elevando o nível de abstração do projetista. Para tanto, foi proposta uma heurística para gerar seqüências parciais de eventos de entrada e saída a partir do modelo de serviço do protocolo de comunicação, especificado através de máquinas de estados finitos, e de um con-

junto de PDUs a ele associado. A correção semântica entre os modelos do serviço, fornecido, e do protocolo, evoluído, foi assegurada através do teste de equivalência inserido no fluxo de execução da metodologia.

A maior vantagem nesse ambiente de projeto refere-se à facilidade de contar com pessoal não especializado, com pouco ou nenhum conhecimento do domínio do problema. O programa que implementou o algoritmo de programação genética na metodologia de síntese de protocolos de comunicação mostrou-se bastante eficiente. Uma execução para um limite máximo de 3000 gerações de uma população de 500 indivíduos consumiu o tempo médio de 15 min utilizando o hardware descrito na seção 3.5. Considerando que as técnicas tradicionais de síntese de protocolo que utilizam máquinas de estados finitos são truncadas para a execução de várias etapas envolvendo transformações em autômatos, o tempo acima mencionado é bastante curto. Estas etapas foram convenientemente reduzidas ou eliminadas pela metodologia apresentada, da seguinte forma: (a) a projeção foi realizada de forma direta; (b) a determinização foi completamente eliminada; e (c) o autômato foi automaticamente reduzido como conseqüência da adoção de um processo evolutivo que contou com a utilização de cromossomos de tamanhos variáveis e com uma função de aptidão ponderada pelo número de estados que implementa o autômato.

Mais recentemente, abordagens destinadas ao projeto de protocolos de comunicação a partir de descrições semiformais, tais como os cenários de interação, têm sido objeto de intensa pesquisa. Sua principal vantagem sobre os métodos de síntese está na especificação inicial do protocolo, que é feita através de representações visuais que favorecem descrições de serviços, mais intuitivas que descrições de agentes. Neste contexto, um método não convencional destinado à geração de protocolos de comunicação a partir de cenários de interação foi proposto neste trabalho. Tal método utiliza técnicas evolutivas e uma heurística para gerar seqüências parciais de eventos de entrada e saída a partir de cenários de interação especificados através de MSCs.

Outra contribuição deste trabalho foi a proposta da utilização de representações baseadas em estados com cromossomo de tamanho variável. Neste contexto, os resultados comprovaram o benefício em se tratar o número de estados durante a evolução do autômato. Tal técnica tem caráter geral, podendo ser empregada diretamente em qualquer

sistema que utilize o formalismo de máquinas de estados finitos. Apesar da representação baseada em estados ser considerada computacionalmente intensiva, por levar em conta todos os eventos de entrada possíveis em cada um dos estados, ela mostrou-se adequada para o desenvolvimento de protocolos de comunicação, pois garante a completude dos mesmos.

Os algoritmos evolutivos são indiscutivelmente eficazes na resolução de problemas complexos, porém algumas técnicas podem ser utilizadas para melhorar ainda mais seus desempenhos. As técnicas de hibridização e as técnicas de manutenção de diversidade da população são algumas das abordagens utilizadas com este objetivo. Neste âmbito, este trabalho contribuiu com a proposta de um método para escape de convergência prematura, de implementação simplificada e custo computacional reduzido, resultando num melhor desempenho do sistema evolutivo. Tal método também tem caráter geral, podendo ser empregado diretamente em qualquer sistema evolutivo que utiliza como formalismo as máquinas de estados finitos.

Como proposta para trabalhos futuros sugere-se estender a metodologia de síntese de circuitos digitais combinacionais incorporando requisitos de performance. A maioria das ferramentas de síntese de alto nível fornece uma medida de atraso para o caminho crítico da estrutura encontrada, por meio de uma função de análise de tempo (*timing analysis*), indicando a freqüência máxima de operação do relógio do sistema. Tal medida pode ser inserida na função de aptidão, direcionando a evolução a buscar soluções com alto desempenho. Nesse caso, pode-se utilizar a estratégia de otimização por agregação de objetivos [77], que obtém uma medida escalar de aptidão a partir de um vetor de objetivos contendo, por exemplo, funcionalidade, área e performance.

O atual estágio de pesquisa do projeto de protocolos de comunicação através de técnicas evolutivas é incipiente. Este trabalho apresentou uma primeira abordagem nesse contexto, que inclui protocolos multi-entidades. Um grande desafio nesta área de pesquisa é prover técnicas para o desenvolvimento de protocolos reais, com maiores números de estados que os admitidos neste trabalho e, possivelmente, que considerem mensagens com parâmetros.

Referências Bibliográficas

[1] SPEARS, W., DE JONG, K., BACK, T. *et al*., "An Overview of Evolutionary Computation", in *Proceedings of the European Conference on Machine Learning*, pp. 442-459, Vienna, Austria, 1993.

[2] DARWIN, C., *On the Origin of Species by Means of Natural Selection*, John Murray, 1859.

[3] BANZHAF, W., NORDIN, P., KELLER, R. *et al*., *Genetic Programming: An Introduction*, San Francisco, CA, Morgan Kaufmann and Heidelberg Publishers, 1998.

[4] ROSCA, J., *Hierarchical Learning with Procedural Abstractions Mechanisms*, Ph.D. thesis, University of Rochester, New York, 1997.

[5] KOZA, J. R., *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, Massachusetts, MIT Press, 1992.

[6] KOZA, J., BENNETT, F., ANDRE, D., *et al*., *Genetic Programming III Videotape: Human Competitive Machine Intelligence*, San Francisco, CA, Morgan Kaufmann, 1999.

[7] ARAÚJO, S., MESQUITA, A., PEDROZA, A., "Using Genetic Programming and High Level Synthesis to Design Optimized Datapath", *in Proceedings of the 5th International Conference of Evolvable Systems (ICES'03)*, LNCS, Vol. 2606, pp. 434-445, Trondheim, Norway, March 2003.

[8] ARAÚJO, S., MESQUITA, A., PEDROZA, A., "Optimized Datapath Design by Evolutionary Computation", *in Proceedings of the 3rd IEEE International Workshop on System-on-Chip for Real-Time Applications (IWSOC'03)*, pp. 06-10, Calgary, Canada, June 2003.

[9] ARAÚJO, S., MESQUITA, A., PEDROZA, A., "Síntese de Circuitos Digitais Otimizados via Programação Genética", *30º Seminário Integrado de Software e Hardware (SEMISH'03)*, Vol. III pp. 273-285, Campinas, Brasil, Agosto de 2003.

[10] ARAÚJO, S., MESQUITA, A., PEDROZA, A., "Improvements in FSM Evolutions from Partial Input/Output Sequences", in *Proceedings of the 1st International Workshop on Evolvable Hardware, (E-HARD'2004)*, LNCS, pp. 1311-1319, Krakow, Poland, June 2004.

[11] ARAÚJO, S., PEDROZA, A., MESQUITA, A., "Evolutionary Synthesis of Communication Protocols", *in Proceedings of the 10th International Conference on Telecommunications (ICT'03)*, pp. 986-993, Tahiti, French Polynesia, February 2003.

[12] ARAÚJO, S., PEDROZA, A., MESQUITA, A., "Uma Metodologia de Projeto de Protocolos de Comunicação Baseada em Técnicas Evolutivas", *20º Simpósio Brasileiro de Telecomunicações (SBT'03)*, Rio de Janeiro, Brasil, Outubro de 2003.

[13] ARAÚJO, S., MESQUITA, A., PEDROZA, A., "A Scenario Based Approach to Protocol Design Using Evolutionary Techniques", in *Proceedings of the 1st European Workshop on Evolutionary Computation in Communications, Networks and Connected Systems (EVOCOMNET'04)*, LNCS, Vol. 3005, pp. 178-187, Coimbra, Portugal, April 2004.

[14] GAJSKI, D., DUTT, N., WU, A., *et al., High Level Synthesis: Introduction to Chip and System Design*, Kluver Academic Publishers, Massachusetts, USA, 1992.

[15] HESSEL, F., MARCHIORO, G., "Concepção Conjunta Hardware/Software (Co-design)", *XVI Jornada de Atualização em Informática, Congresso da Sociedade Brasileira de Computação*, pp. 449-494, 1997.

[16] LUGER, G., STUBBLEFIELD, W., *Artificial Intelligence: Structures and Strategies for Complex Problem Solving*, Addison-Wesley, 3rd edition, 1998.

[17] SIMON, H., *Why Should Machines Learn*, Palo Alto, in Michalski *et al*. ed., 1983.

[18] NEWELL, A., *Physical Symbol Systems*, in Norman, 1981.

[19] NEWELL, A., SIMON, H., "Computer Science as Empirical Inquiry: Symbols and Search", *Communications of the ACM*, Vol. 19, No 13, pp. 113-126. 1976.

[20] FOGEL, L., *Autonomous Automata*, Industrial Research, Vol. 4, pp. 14-19, 1962.

[21] FOGEL, L., OWENS, A., WALSH, M., *Artificial Intelligence Through Simulated Evolution*, John Wiley & Sons, New York, 1966.

[22] HOLLAND, J., *Adaptation in Natural and Artificial Systems*, 1st ed., University of Michigan Press, Ann Arbor, 1975.

[23] GOLDBERG, D., *Genetic Algorithm in Search, Optimization and Machine Learning*, Addison-Wesley Publishing, Massachusetts, 1989.

[24] RECHENBERG, I., "Evolutionsstrategie: Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution", Frommann-Holzboog, Stuttgart, 1973.

[25] SCHWEFEL, H., *Numerical Optimization of Computer Models*, New York, John Wiley & Sons, 1981.

[26] HOLLAND, J., *Escaping Brittleness: The Possibilities of General Proposed Learning Algorithms Applied to Parallel Rule-Based Systems*, San Mateo, in Michalski *et al*. ed., 1986.

[27] GOLDBERG, D., RICHARDSON, J., "Genetic Algorithm with Sharing for Multimodal Function Optimization", *in Genetic Algorithm and their Applications (ICGA'87)*, Grefenstette J.J. ed., 41-49, 1987.

[28] MIRANDA, M., *Uma Metodologia de Hardware/Software CoDesign de Protocolos de Comunicação Baseada na Otimização de Desempenho por Algoritmos Genéticos*, Tese de D.Sc., COPPE-UFRJ, Rio de Janeiro, Brasil, 2002.

[29] COELLO, C., AGUIRRE, A., BUCKLES, B., "Evolutionary Multiobjective Design of Combinational Logic Circuits", in *Proceedings of the 2nd NASA/DoD Workshop on Evolvable Hardware*, pp. 161-170, IEEE Computer Society, Los Alamitos, California, 2000.

[30] WHIGHAM, P., "Inductive Bias and Genetic Programming", *in 1st International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications*, pp. 461-466, UK:IEE, 1995.

[31] NORDIN, P., "A Compiling Genetic Programming System that Directly Manipulates Machine Code", in *Kinnear, Jr. editor, Advances in Genetic Programming*, chapter 14, pp. 311-331, Cambridge, MA, MIT Press, 1994.

[32] GRUAL, F., "Genetic Synthesis of Boolean Neural Networks with a Cell Rewriting Developmental Process", in *Schaffer and Whitley editors, Proceedings of the*

Workshop on Combinations of Genetic Algorithms and Neural Networks, pp. 55-74, IEEE Computer Society Press, Baltimore, MD, 1992.

[33] TELLER, A., VELOSO, M., *PADO: Learning Tree Structured Algorithms for Orchestration into an Object Recognition System*, Technical Report CMU-CS-95-101, Carnegie Mellon University, Pittsburgh, PA, 1995.

[34] IBA, H., SATO, T., DEGARIS, H., "Numerical Genetic Programming for System Identification", *in Proceedings of the Workshop on Genetic Programming*, Rosca, J. P. editor, pp. 64-75, Tahoe City, California, 1995.

[35] WHIGHAM, P., "Grammatically-Based Genetic Programming", *in Proceedings of the Workshop on Genetic Programming: From The Theory to Real-World Applications*, Morgan Kaufmann, pp. 33-41, 1995.

[36] HÖRNER, H., "A C++ Class Library for Genetic Programming", *Release 1.0 Operating Instructions*, Viena University of Economy, Viena, 1996.

[37] KELLER, R., BANZHAF, W., "Genetic Programming Using Genotype-Phenotype Mapping from Linear Genomes into Linear Phenotypes", *in Proceedings of Genetic Programming 1996*, pp. 116-122, MIT Press, 1996.

[38] BANZHAF, W., "Genotype-Phenotype Mapping and Neutral Variation: A Case Study in Genetic Programming", *in Y. Davidor et al. (eds.), Parallel Problem Solving from Nature III*, pp. 322-332, Berlin, 1994.

[39] WONG, M., LEUNG, K., "Applying Logic Grammars to Induce Sub-Functions in Genetic Programming", *in Proceedings of 1995 IEEE Conference on Evolutionary Computation*, pp. 737-740, USA:IEEE Press, Perth, Australia, 1995.

[40] PATERSON, N., LIVESEY, M., "Evolving Cache Algorithms in C by GP", *in Genetic Programming 1997*, pp. 262-267, MIT Press, 1997.

[41] RYAN, C., O'NEILL, M., "Grammatical Evolution: Evolving Programs for an Arbitrary Language", Lecture Notes in Computer Science 1391, *in Proceedings of the 1st European Workshop on Genetic Programming*, Springer-Verlag, pp. 83-96, Paris, 1998.

[42] DONOVAN, J., *Systems Programming*, Massachusetts, McGraw-Hill, 1972.

[43] RYAN, C., O'NEILL, M., "Grammatical Evolution: A Steady State Approach", *In Proceeding of the 2nd International Workshop on Frontiers in Evolutionary Algorithms*, pp. 180-185, Edinburgh, 1998.

[44] WHIGHAM, P., "Search Bias, Language Bias and Genetic Programming", *in Genetic Programming 1996: Proceedings of the First Annual Conference*, MIT Press, pp. 230-237, 1996.

[45] KOZA, J., *Genetic Programming II: Automatic Discovery of Reusable Programs*, MIT Press, Cambridge, 1994.

[46] RATLE, A., SEBAG, M., "Genetic Programming and Domain Knowledge: Beyond the Limitations of Grammar-Guided Machine Discovery", *PPSN VI*, pp. 211-220, Paris, France, 2000.

[47] CHOMSKY, N., *Aspects of the Theory of Syntax*, Cambridge, MA, MIT Press, 1965.

[48] KIMURA, M., *The Neutral Theory of Molecular Evolution*, Cambridge University Press, 1983.

[49] MUKAI, T., "Experimental Verification of the Neutral Theory", in T. Ohta *et al*. (eds.), *Population Genetics and Molecular Evolution*, Berlin, 1985.

[50] O'NEILL, M., RYAN, C., "Automatic Generation of High Level Function Using Evolutionary Algorithms", *in Proceedings of SCASE 1999, Soft Computing and Software Engineering Workshop*, University of Limerick, Ireland, 1999.

[51] ANGELINE, P., "Genetic Programming and Emergent Intelligence, *In Advances in Genetic Programming*, K. Kinnear (ed.), pp. 75-98, Cambridge, MA, MIT Press, 1994.

[52] WOLPERT, D., MACREADY, W., *No-Free Lunch Theorem for Search*, Technical Report SFI-TR-95-02-010, Santa Fe Institute, NM, 1995.

[53] NAVABI, Z., *VHDL: Analysis and Modeling of Digital Systems*, Massachusetts, McGraw Hill, 2nd edition, 1998.

[54] DEGARIS, H., *Evolvable Hardware: Genetic Programming of a Darwin Machine*, McGraw-Hill, in Artificial Neural Nets and Genetic Algorithms, Springer Verlag,

New York, 1993.

[55] ZEBULUM, R., PACHECO, M. A., VELLASCO, M., "Evolvable Systems in Hardware Design: Taxonomy, Survey and Applications", *in Evolvable Systems: From Biology to Hardware, (ICES96)*, pp. 344-358, Tsukuba, Japan, 1996.

[56] LIN, Y-L, "Recent Development in High Level Synthesis", *ACM Transactions on Design Automation of Electronic Circuits*, Vol. 2, No 1, 1997.

[57] BENNETT III, F., KOZA, J., YU, J., *et al*., "Automatic Synthesis, Placement and Routing of an Amplifier Circuit by Means of Genetic Programming", *in Proceedings of the 3rd International Conference on Evolvable Systems, ICES 2000*, pp. 1-10, Edinburgh, Scotland, UK, 2000.

[58] DAMIANI, E., TETTAMANZI, A., LIBERALI, V., "Automatic Synthesis of Hashing Functions Circuits using Evolutionary Techniques", *XI Brazilian Symposium on Integrated Circuit Design (SBCCI98)*, pp. 42-45, Búzios, Rio de Janeiro, 1998.

[59] MILLER, J., THOMSON, P., FOGARTY, T., "Designing Electronic Circuits Using Evolutionary Algorithms. Arithmetic Circuits: A Case Study", *in Genetic Algorithms and Evolution Strategies in Engineering and Computer Science*, D. Quagliarella *et al*. (eds.), Publisher: Wiley, 1997.

[60] KALGANOVA, T., MILLER, J., "Evolving More Efficient Digital Circuits by Allowing Circuit Layout Evolution and Multi-Objective Fitness", *in Proceedings of the First NASA/DoD Workshop on Evolvable Hardware*, pp. 54-63, Los Alamitos, CA, IEEE Computer Society Press, 1999.

[61] HOUNSELL, B., ARSLAN, T., "A Novel Evolvable Hardware Framework for the Evolution of High Performance Digital Circuits", *GECCO-2000*, pp. 525-532, Las Vegas, NV, 2000.

[62] THOMPSON, A., "An Evolved Circuit, Intrinsic in Silicon, Entwined with Physics", *in Proceedings of the 1st International Conference on Evolvable Systems: From Biology to Hardware, ICES96*, pp. 390-405, Tsukuba, Japan, 1996.

[63] MONTANA, D., POPP, R., IYER, S. *et al.*, "EvolvaWare: Genetic Programming for Optimal Design of Hardware-Based Algorithms", *Proceedings of the Third An-*

nual Genetic Programming Conference, pp. 869-874, Madison, Wisconsin, 1998.

[64] HEMMI, H., MIZOGUCHI, J., SHIMOHARA, K., "Development and Evolution of Hardware Behaviors", *Towards Evolvable Hardware: The Evolutionary Engineering Approach, International Worksho*p, Lausanne, edited by E. Sanchez and M. Tomassini, Springer-Verlag, LNCS 1062, pp. 250-265, 1995.

[65] GAJSKI, D., TADATOSHI, I., CHAIYAKUL, V., *et al*., *A Design Methodology and Environment for Interactive Behavioral Synthesis*, Technical Report 96-29, University of Irvine, CA, 1996.

[66] GAJSKI, D., ZHU, J., DÖRNER, R., *Special Issues in Codesign*, Technical Report ICS-97-26, University of Irvine, CA, 1997.

[67] THOMPSON, A., "Exploration in Design Space, Unconventional Electronics Design Through Artificial Evolution", *IEEE Transactions in Evolutionary Computation*, Vol. 3, No 2, pp. 167-196, 1999.

[68] IEEE Std. 1076-1993, *VHDL Language Reference Manual*, 1993.

[69] KUUSILINNA, K., HÄMÄLÄINEN, T., SAARINEN, J., *Practical VHDL Optimization for Timing Critical FPGA Applications*, Microprocessors and Microsystems 23, pp. 459-469, 1999.

[70] LANDIS, D., *Programmable Logic and Application Specific Integrated Circuits*, Handbook of Components for Electronics, Chapter II, Vol. I, 1995.

[71] SYNOPSYS Inc., *Online Documentation*, http://www.synopsys.com, 2001.

[72] ALTERA Inc., *Altera Digital Library Databook 2001*, 2001.

[73] ZEBULUM, R., PACHECO, M. A., VELLASCO, M., "Evolutionary Synthesis Applied to the Synthesis of a CPU Controller", *in Lecture Notes in Artificial Intelligence, Springer-Verlag, 2nd Asia-Pacific Conference on Simulated Evolution and Learning (SEAL98)*, Vol. 1585, Camberra, Australia, 1998.

[74] MANOVIT, C., APORNTEWAN, C., CHONGSTITVATANA, P., "Synthesis of Synchronous Sequential Logic Circuits from Partial Input/Output Sequences", *in Proceedings of International Conference of Evolvable Systems (ICES'98)*, pp. 98-105, Lausanne, Switzerland, 1998.

[75] SENTOVICH, E., SINGH, K., MOON, C., *et al*., "Sequential Circuit Design Using Synthesis and Optimization", *in Proceedings of the IEEE International Conference on Computer Design*, pp. 328-333, Cambridge, MA, 1992.

[76] SYSTÄ, T., *Static and Dynamic Reverse Engineering Techniques for Java Software Systems*, Ph.D. thesis, University of Tampere, Finland, 2000.

[77] ZEBULUM, R., *Síntese de Circuitos Eletrônicos por Computação Evolutiva*, Tese de D.Sc., PUC-Rio, Rio de Janeiro, Brasil, 1999.

[78] COLLINS, R., JEFFERSON, D., "Representation for Artificial Organisms", *in Proceedings of the 1st* International Conference on Simulation of Adaptive Behavior, MIT Press, Paris, 1991.

[79] CHONGSTITVATANA, P., APORNTEWAN, C., "Improving Correctness of Finite-State Machine Synthesis from Multiple Partial Input/Output Sequences", *in Proceedings of the 1st NASA/DoD Workshop on Evolvable Hardware*, pp. 262-266, Pasadena, CA, 1999.

[80] CLANCY, D., KUIPERS, B., "Model Decomposition and Simulation: A Component Based Qualitative Simulation Algorithm", *in Proceedings of* the 14th *National Conference on Artificial Intelligence*, AAAI Press, Rhode Island, 1997.

[81] ERCEGOVAC, M., LANG, T., MORENO, J., *Introduction to Digital Systems*, Publisher: Wiley, 1999.

[82] FELLER, W., *An Introduction to Probability Theory and its Applications*, Vol. I, Wiley, pp. 224-225, 1968.

[83] SPEARS, W., GORDON, D., "Evolving Finite-State Machine Strategies for Protecting Resources", *in Proceedings of ISMIS'00*, pp. 166-175, Charlotte, North Carolina, 2000.

[84] URSEM, R., "Diversity-Guided Evolutionary Algorithms", *in Proceedings of Parallel Problem Solving from Nature VII (PPSN-2002)*, pp. 462-471, 2002.

[85] KHARE, V., *Artificial Speciation and Automatic Modularization*, MSc. dissertation, University of Birmingham, Birmingham, 2002.

[86] DEJONG, K., *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*,

Ph.D. thesis, University of Michigan, Ann Arbor, MI, 1975.

[87] MAHFOLD, S., "Crowding and Preselection Revisited", *in Proceedings of Parallel Problem Solving from Nature (PPSN-II)*, North-Holland, pp. 27-36, 1992.

[88] SMITH, R., FORREST, S., PERELSON, A., "Searching for Diverse, Cooperative Populations with Genetic Algorithms", *Evolutionary Computation 1993*, Vol. 1, No. 2, pp. 127-149, 1993.

[89] LANGDON, W., *Evolution of Genetic Programming Populations*, University College London Technical Report RN/96/125, 1996.

[90] HIROAKI, U., NORIYUKI, I., KENICHI, T. *et al.*, "Acquisition of a State Transition Graph using Genetic Network Programming Techniques", 2001.

[91] HOLZMANN, G., *Design and Validation of Computer Protocols*, Prentice Hall Int., New Jersey, 1991.

[92] KHOUMSI, A., SALEH, K., "Two Formal Methods for the Synthesis of Discrete Event Systems", *Computer Networks and ISDN Systems*, Vol. 29, pp. 759-780, 1997.

[93] PROBERT, R., SALEH, K., "Synthesis of Protocols: Survey and Assessment", *IEEE Transactions on Computers*, Vol. 40, No. 4, pp. 468-476, 1991.

[94] TANENBAUM, A., *Computer Networks*, 3rd edition, New Jersey, Prentice Hall, 1996.

[95] CHU, P. Y. M., LIU, M. T., "Synthesizing Protocol Specifications from Services Specifications in FSM Model", *in Proceedings of the IEEE Computer Networking Symposium*, pp. 173-182, Washington DC, 1988.

[96] SALEH, K., PROBERT, R., "Automatic Synthesis of Protocol Specifications from Service Specifications", *in Proceedings of the 10th International Phoenix Conference on Computers and Communications*, IEEE Computer Society Press, Phoenix, 1991.

[97] YAMAGUCHI, H., OKANO, K., HIGASHINO, T., *et al.*, "Synthesis of Protocol Entities' Specifications from Service Specifications in a Petri Net Model with Registers", *in Proceedings of ICDCS*-15, pp. 510-517, 1995.

[98] PARK, J-C., MILLER, R., "Synthesizing Protocol Specifications from Service Specifications in Timed Extended Finite State Machines", *International Conference on Distributed Computing Systems*, 1996.

[99] KHOUMSI, A., BOCHMANN, G., DSSOULI, R., "Protocol Synthesis for Real-Time Applications", *in Proceedings of the FORTE*, pp. 417-433, 1999.

[100] YAMAGUCHI, H., *Implementation of Service Specifications on Distributed Computing Systems*, Ph.D. thesis, Osaka University, Osaka, Japan, 1998.

[101] EL-FAKIH, K., YAMAGUCHI, H., BOCHMANN, G., *et al*., "Protocol Re-synthesis Based on Extended Petri Nets", *in Proceedings of International Workshop on Software Engineering and Petri Nets (SEPN-2000)*, pp. 173-188, 2000.

[102] EL-FAKIH, K., YAMAGUCHI, H., BOCHMANN, G., "A Method and a Genetic Algorithm for Deriving Protocols for Distributed Applications with Minimum Communication Cost", *in Proceedings of the 11th IASTED*, (PDCS'99), Cambridge, USA, 1999.

[103] CLEAVELAND, R., PARROW, J., STEFFEN, B., "The Concurrency Work-bench: A Semantics Based Toll for the Verification of Concurrent Systems", *ACM Transactions on Programming Languages and Systems*, 15(1):36-72, 1994.

[104] MILNER, R., *Communication and Concurrency*, Prentice Hall, New Jersey, 1989.

[105] PARROW, J., "Verifying a CSMA/CD-Protocol with CCS", *In Proceedings of the 8th Symposium on Protocol Specification, Testing and Verification, IFIP*, pp. 373-387, Amsterdam, 1996.

[106] BOCHMANN, G., PETRENKO, A., "Protocol Testing: A Review of Methods and Relevance for Software Testing", *ISSTA'94*, pp. 109-124, Seattle, 1994.

[107] TAI, K., YOUNG, Y., "Synchronizable Test Sequences of Finite State Machines", *Computer Networks and ISDN Systems*, Vol. 30, pp. 1111-1134, 1995.

[108] KUMAR, B., VENKATARAM, P., "An Optimization Technique for Protocol Conformance Test Sequence Generation Based on MUIOS Using Hopfield Neural Networks", pp. 459-469, Kharagpur, India, 1996.

[109] SIDHU, D., LEUNG, T., "Formal Methods for Protocol Testing: A Detailed

Study". *IEEE Transactions on Software Engineering*, Vol. SE-15, pp. 413-426, April 1989.

[110] AMYOT, D., EBERLEIN, A., "An Evaluation of Scenario Notations and Construction Approaches for Telecommunication Systems Development", *Telecommunications Systems Journal*, 24:1, pp. 61-94, September 2003.

[111] AMYOT, D., *Specification and Validation of Telecommunication Systems with Use Case Maps and LOTOS*, Ph.D. thesis, University of Ottawa, Ottawa, Canada, 2001.

[112] KRÜGER, I., *Distributed System Design with Message Sequence Charts*, Ph.D. thesis, Technische Universität München, München, Germany, 2000.

[113] KOSKIMIES, K., MÄNNISTÖ, T., SYSTÄ, T. *et al*., *Automated Support for Modeling OO Software*, IEEE Software, Vol. 15, No. 1, pp. 87-94, 1998.

[114] ALUR, R., ETESSAMI, K., YANAKAKIS, M., "Inference of Message Sequence Charts", *in 22nd International Conference on Software Engineering (ICSE'00)*, Lemeric, Ireland, 2000.

[115] ANGLUIN, D., SMITH, C., "Inductive Inference, Theory and Methods", *ACM Computer Survey*, pp. 237-269, 1983.

[116] UCHITEL, S., KRAMER, J., MAGEE, J., "Detecting Implied Scenarios in Message Sequence Chart Specification", Dept. of Computing, Imperial College, 2001.

[117] KOSKIMIES, K., MÄKINEN, E., *Automatic Synthesis of State Machines from Trace Diagrams*, Software Practice and Experience, Vol. 24, No. 7, pp. 643-658, 1994.

[118] PETRENKO, A., BOCHMANN, G., "On Fault Coverage of Tests for Finite State Specifications", *Computer Networks ISDN Systems*, Vol. 29, No. 1, pp. 81-106, 1996.

[119] STEVENS, P., *The Edinburgh Concurrent Workbench User Manual – Version 7.1*, available at http://www.dcs.ed.ac.uk/home/cwb/doc, 1999.

Apêndice A

CWB (Concurrency Workbench)

Uma Ferramenta Baseada em Semânticas para a Verificação de Sistemas Concorrentes

O CWB (*Concurrency Workbench*) é uma ferramenta automática para a análise de redes de processos de estados finitos expressas através de CCS. Essa ferramenta permite examinar sistemas concorrentes através de uma variedade de métodos de verificação para diferentes modelos semânticos. Em particular, o CWB realiza vários testes de equivalência e checagem de modelos (*model checking*), que são utilizados para decidir se um sistema apresenta o comportamento desejado. Dentre outras competências, o CWB permite [119]:

- Definir comportamentos na sintaxe de CCS com semântica de temporização (TCCS: *Temporal CCS*) ou CCS Síncrono (SCCS: *Synchronous CCS*) e realizar várias análises nesses comportamentos, tais como a análise do espaço de estados de um processo ou a checagem de equivalências semânticas;
- Definir proposições através de uma lógica modal poderosa e checar quando um processo satisfaz a especificação formulada nessa lógica;
- Derivar automaticamente fórmulas lógicas que distinguem processos não equivalentes;
- Simular interativamente o comportamento de um agente, guiando-o através de seu espaço de estados de maneira controlada.

O CWB foi desenvolvido pela Universidade de Edinburgh e encontra-se na versão 7.1. Sob o ponto de vista do usuário, o CWB é composto por um módulo Básico, que constitui o núcleo do sistema, e por diversos módulos opcionais que podem ser incluídos durante sua instalação. O CWB está disponível para várias plataformas. A seção A.5 mostra os procedimentos para a obtenção e instalação do CWB v7.1 em ambiente Windows.

Este anexo encontrasse estruturado como se segue. A seção A.1 apresenta o formato básico e a sintaxe do CWB. A seção A.2 lista os combinadores do TCCS. Os comandos do CWB são detalhados na seção A.3. A seção A.4 apresenta um exemplo de especificação em CCS. Por último, a seção A.5 lista os passos para a instalação do CWB v.7.1 em ambiente Windows.

A.1 Formato Básico e Sintaxe de Entrada

A.1.1 Fundamentos da Sintaxe

Todos os comandos do CWB terminam com “;” seguido por “nova linha”. Qualquer caractere após o ponto e vírgula será ignorado. Comandos e definições de agentes podem ocupar várias linhas, não sendo permitido, entretanto, colocar mais de um comando em uma única linha, mesmo se separados por “;”. Qualquer caractere posicionado entre o caractere “*” e uma nova linha será considerado comentário. “*” e “;” não poderão ser utilizados em *strings* , incluindo nomes de arquivos e argumentos.

A.1.2 Variáveis e Identificadores

Nomes de ações e parâmetros formais de ações iniciam com letra minúscula ($a - z$). Identificadores para conjuntos, funções de re-rotulação, agentes ou proposições e seus

parâmetros formais iniciam com letra maiúscula (*A – Z*). O segundo e os caracteres seguintes podem ser:

- Letras minúsculas (*a – z*);
- Letras maiúsculas (*A – Z*);
- Dígitos (0 – 9);
- Os caracteres ? ! _ ‘ ’ -;
- Qualquer caractere entre pares de símbolos “%”.

eps e *tau* não podem ser utilizados como nomes de ações ou parâmetros formais de ações e *T* e *F* não podem ser utilizados como identificadores de proposições ou parâmetros formais de proposições.

A.2 O TCCS

O CCS com semântica de temporização (TCCS) é a linguagem *default* de entrada do CWB. A tabela A.1 apresenta as ações e os combinadores básicos do TCCS, excluindo as extensões de temporização que não foram utilizadas neste trabalho.

Tabela A.1: Ações e combinadores básicos do TCCS.

AÇÕES (a)	**DESCRIÇÃO**
tau	Ação interna
a	Nome: um identificador iniciando com letra minúscula
‘a	CoNome: denota *a* barrado
eps	Observação vazia
AGENTES (A)	**DESCRIÇÃO**
0	Deadlock
a.A	Prefixo de ação

A + *A*	Escolha fraca (*weak choice*)
A + + *A*	Escolha forte (*strong choice*)
A | *A*	Composição paralela
A \ a	Restrição de uma simples ação
A \ *S*	Restrição de um conjunto
A[*R*]	Re-rotulação

A.3 Comandos do CWB

Nessa seção será apresentada uma lista dos principais comandos disponíveis no sistema CWB v7.1, com uma breve descrição de suas operações.

A.3.1 Utilizando Ajuda On-line

Comandos:

help: apresenta uma panorama da ajuda no CWB.

help commands: lista todos os comandos, seguido de uma breve descrição de cada comando.

help (command): descreve especificamente o comando solicitado.

A.3.2 Saindo do Módulo Básico

Comando:

quit: termina uma seção do CWB (comando alternativo: **bye**).

A.3.3 Definindo Identificadores e Mantendo o Ambiente

Comandos:

agent X = A: associa o agente *A* ao identificador X.

agent X: imprime a definição do agente X.

set S = a,b,...: associa um conjunto de ações ao identificador de conjunto S.

print: imprime a definição de todos os identificadores.

clear: remove todos as associações.

A.3.4 Trabalhando com Arquivos

Comandos:

input "file": executa o CWB sobre o arquivo onde encontra-se a descrição de um sistema em CCS. O arquivo pode conter qualquer comando CWB, exceto o comando **sim** (simulação interativa). Se algum erro for detectado o CWB terminará sua execução retornando ao console padrão.

output "file": controla quando a saída do CWB é escrita sobre um dado arquivo. O CWB mantém uma pilha de saída de forma a permitir comandos de saída aninhados.

save "file": salva o contexto corrente no arquivo indicado.

A.3.5 Trabalhando com um Único Agente

Os seguintes comandos requerem um único agente como argumento.

Comandos:

prefixform A: imprime um agente na forma de prefixo.

transitions A: lista, passo a passo (*single step*), as transições de um agente.

graph A: lista o grafo de transições de um agente.

size A: fornece o número de estados de um agente (com número finito de estados).

closure (a,A): lista os agentes alcançáveis a partir de *A*, através da ação *a*.

deadlocks A: lista os estados que apresentam *deadlock* e *livelock* e os traços que levam a esses estados.

derivatives (a,A) lista as derivações de um agente a partir da ação *a*.

freevars A: lista as variáveis livres (ou re-rotuladas) de um agente.

init (a,b,...,A): lista as ações observáveis que um agente pode realizar imediatamente.

obs (n,A): lista as ações de um dado comprimento *n* de um agente e o estado final de cada uma destas ações.

random (n,A): lista as ações pseudo randômicas de comprimento máximo *n*.

stable A: informa se o agente é estável, isto é, se ele realiza apenas ações observáveis (ou seja, não realiza ações não observáveis).

states A: lista o espaço de estados de um agente (com número finito de estados).

statesobs A: lista o espaço de estados de um agente, assim como as ações observáveis pelas quais os estados podem ser alcançados a partir do estado inicial.

A.3.6 Comparando Dois Agentes

Os seguintes comandos requerem dois agentes como argumento. Nenhum desses comandos finaliza se um dos agentes tem espaço de estados infinito.

Comandos:

cong (A,B): retorna *true* em caso de congruência (igualdade) observacional.

diveq (A,B): retorna *true* em caso de divergência.

eq (A,B): retorna *true* em caso de equivalência observacional (equivalência fraca).

strongeq (A,B): retorna *true* em caso de equivalência observacional (equivalência forte).

A.3.7 Simulando o Comportamento de um Agente

O comando **sim** simula um agente interativamente. Os seguintes comando estão disponíveis dentro do simulador.

Comandos:

sim A: simula o agente *A*.

menu: lista as transições, passo a passo, a partir do estado corrente.

random n: simula até *n* passos, selecionando transições aleatoriamente.

history: lista os estados e as transições pelas quais o estado corrente é alcançado.

break a,b…: estabelece pontos de parada em ações *a*, *b*, etc..

lb: lista todos os pontos de parada.

db a,b…: exclui *a*, *b*, etc., da lista de pontos de parada.

help: imprime um sumário dos comandos de simulação interativa.

quit: termina a simulação e retorna ao *prompt* do CWB.

A.4 Utilizando o CWB

O CWB é um sistema interativo: quando invocado, o usuário poderá emitir comandos que irão, tipicamente, associar indicadores, analisar derivações de processos ou, principalmente, checar equivalências entre processos.

A listagem abaixo descreve uma implementação em CCS do modelo de serviço do protocolo orientado à conexão (agente *Servico_OC*), das entidades de protocolo (agentes Emissor e Receptor) e a composição entre elas (agente *OC*). Em seguida, o CWB é invocado para realizar o teste de equivalência entre os dois modelos (*OC* e *Servico_OC*). O resultado deste teste (*true*, no caso) indica que os dois agentes são, de fato, observacionalmente equivalentes.

```
**** Protocolo de Abertura e Fechamento de Conexão****

*** Conjunto de Interações***
set Internos = {con_req,con_acc,rei_req,rei_acc,dis_req,dis_acc};

*** Sistema Emissor/Receptor***
agent OC = (Emissor|Receptor)\Internos;
```

```
***Comportamento Emissor***
agent Emissor = conReq.'con_req.Esp_CC;
agent Esp_CC = con_acc.'conConf.Dt_I;
agent Dt_I = reiReq.'rei_req.Esp_RA;
agent Esp_RA = rei_acc.'reiConf.Dt_S;
agent Dt_S = disReq.'dis_req.Esp_DC;
agent Esp_DC = dis_acc.Emissor;

***Comportamento do Receptor***
agent Receptor = con_req.'conInd.Esp_Resp;
agent Esp_Resp = conResp.'con_acc.Dt_RR;
agent Dt_RR = rei_req.'reiInd.Esp_Rei;
agent Esp_Rei = reiResp.'rei_acc.Dt_R;
agent Dt_R = dis_req.'disInd.'dis_acc.Receptor;

***Especificacao do Servico***
agent Servico_OC = con-
Req.'conInd.conResp.'conConf.reiReq.'reiInd.reiResp.'reiConf.disReq.'d
isInd.Servico_OC;

*** ----------------------

command: eq (OC,Servico_OC)
true
```

A.5 Instalando o CWB V7.1 em Ambiente Windows

Para executar o CWB no sistema operacional Windows será necessário compilá-lo, utilizando o compilador *Standard ML* de New Jersey. O processo de obtenção de programas em sites oficiais, de compilação e de instalação, seguem os seguintes passos:

- Na página http://www.dcs.ed.ac.uk/home/cwb/getting.html, escolher a álgebra de processos de interesse (TCCS ou SCCS). Neste trabalho foi utilizado o TCCS;
- Nessa mesma página selecionar os módulos opcionais que são apresentados em uma tabela. Para realizar a checagem de equivalência será necessário incluir o módulo *equivalences*;
- Baixar o arquivo *custom.ml*, construído com os dados fornecidos nos passos anteriores, pressionando-se o botão “Download Custom Build File” disponível ao final da tabela. Salvá-lo no diretório do c:\sml\bin;

- Baixar o arquivo *smlnj.exe* da página http://cm.bell-labs.com/cm/cs/what/smlnj e instalá-lo no diretório c:\sml;
- Executar o smlnj e invocar o comando *-use "c:\\sml\\bin\\custom.ml;"*, que produzirá o arquivo *cwb.x86-win32*;
- O CWB é executado a partir da linha de comando. No diretório c:\sml\bin (prompt do DOS), invocar o comando *sml @SMLload=cwb.x86-win32*. A partir desse momento o CWB já está ativado. Para carregar uma especificação utilizar o comando *input "nome_arquivo.ccs;"*, onde nome_arquivo.ccs é uma descrição em CCS.