Verificação e Síntese de Sistemas Híbridos

Adilson Luiz Bonifácio

Dissertação de Mestrado

Instituto de Computação
Universidade Estadual de Campinas

Verificação e Síntese de Sistemas Híbridos

Adilson Luiz Bonifácio[1]

Março de 2000

Banca Examinadora:

- Prof. Dr. Arnaldo Vieira Moura (Orientador)
- Prof. Dr. João Batista Camargo Jr.
  Escola Politécnica, USP
- Prof. Dr. Guido Costa Souza de Araújo
  Intituto de Computação, Unicamp
- Prof. Dr. Luiz Eduardo Buzato (Suplente)
  Intituto de Computação, Unicamp

[1]Suporte parcial da CAPES, DS-44/97.

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP

Bonifácio, Adilson Luiz

B641v Verificação e síntese de sistemas híbridos / Adilson Luiz Bonifácio -- Campinas, [S.P. :s.n.], 2000.

Orientador : Arnaldo Vieira Moura

Dissertação (mestrado) - Universidade Estadual de Campinas, Instituto de Computação.

1. Linguagens formais. 2. Modelagem. I. Moura, Arnaldo Vieira. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Verificação e Síntese de Sistemas Híbridos

Este exemplar corresponde à redação final da Dissertação devidamente corrigida e defendida por Adilson Luiz Bonifácio e aprovada pela Banca Examinadora.

Campinas, 28 de abril de 2000.

Prof. Dr. Arnaldo Vieira Moura (Orientador)

Dissertação apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

TERMO DE APROVAÇÃO

Tese defendida e aprovada em 28 de abril de 2000, pela Banca Examinadora composta pelos Professores Doutores:

Prof. Dr. João Batista de Camargo Júnior
USP

Prof. Dr. Guido Costa Souza de Araújo
IC-UNICAMP

Prof. Dr. Arnaldo Vieira Moura
IC-UNICAMP

Resumo

Sistemas distribuídos híbridos advêm da interconexão de sistemas de dinâmica contínua com sistemas de dinâmica discreta. A noção de autômatos híbridos oferece meios para a construção de especificações formais para tais sistemas. Autômatos híbridos são autômatos finitos, onde cada estado descreve um perfil dinâmico do sistema e cujas transições entre estados provocam alterações nestes perfis dinâmicos.

Neste trabalho, alguns sistemas híbridos reais são modelados através de autômatos híbridos. Em seguida, os modelos construídos são verificados, usando-se das facilidades de uma ferramenta computacional. Além disso, alguns parâmetros importantes, que afetam o comportamento operacional dos modelos, têm seus valores sintetizados.

Os sistemas alvo desse trabalho são segmentos de via de uma malha metroviária e um sistema de gerenciamento de tráfego aéreo. As verificações foram sempre conduzidas de maneira a garantir uma operação segura dos sistemas estudados. As sínteses realizadas contribuíram para determinar valores mais justos para os parâmetros operacionais enfocados, mantendo a segurança na operação dos sistemas alvo.

Abstract

Distributed hybrid systems result from the interplay of continuous and discrete dynamics systems. The notion of hybrid automata offers a way to formally specify such systems. A hybrid automaton is a finite state automaton, where each state is extended to contain a description for a system dynamic profile. Transitions between states model a change in the system dynamics.

In this work, some real hybrid systems are modeled using the formalism of hybrid automata. Next, the models constructed are verified, using the support of a computational tool. Moreover, values are synthesized for some important parameters that affect the system operational behavior.

The target systems treated here are segments of a subway mesh and an air traffic control system. The verification sessions aimed at certifying that the system operates safely. Results from the synthesis contributed to obtain tighter values for operational system parameters, while still guaranteeing its safe operation.

Agradecimentos

Em primeiro lugar gostaria, como não poderia deixar de ser, de agradecer a Deus, que sempre esteve ao meu lado, me deu saúde e força para superar as dificuldades e prosseguir minha caminhada. Agradeço-Lhe por me acalmar quando estive angustiado, por me confortar nos momentos difíceis, por me alegrar quando me entristeci, por iluminar meu caminho sempre que me desviei e por permanecer no meu coração. "O Senhor é meu Pastor, nada me faltará".

Agradeço imensamente aos meus Pais, Alexandre Bonifácio e Setsuko Higa Bonifácio, responsáveis pelos passos que segui na minha vida, me incentivando e me dando apoio a todo momento. Agradeço por terem me ensinado dignidade, honestidade e perseverança, e a sempre seguir as aspirações que almejei para minha vida. Por terem confiado e acreditado na minha pessoa. Pelo amor e compreensão que sempre me deram. Por isso, não apenas agradecer, gostaria de dedicar-lhes mais essa vitória. Muito obrigado aos meus queridos Pais.

Aos meus irmãos, Alex, Ailton e Carla, por terem sempre me passado palavras de otimismo, apoio e esperança.

Agradeço principalmente ao meu orientador, Prof. Dr. Arnaldo Vieira Moura, pela dedicação e pela atenção dispensadas. Pela seriedade durante todo o tempo de realização do trabalho, e por tudo que aprendi ao longo dessa convivência.

Agradeço a todos os meus amigos, de longa data, pela grande amizade e companheirismo, e também aos novos amigos que conquistei nesse período.

Aos professores e funcionários do Instituto de Computação, que direta ou indiretamente, tenham contribuído para meu crescimento pessoal, cultural e intelectual.

A todos meu muito obrigado.

Se o mundo está equivocado, lembra-te que nele há criaturas como tu.

Gandhi

Nunca ande pelo caminho traçado, pois ele conduz somente até onde os outros já foram.

Graham Bell

...E nunca considerem seu estudo como uma obrigação, mas sim como uma oportunidade invejável de aprender, sobre a influência libertadora da beleza no domínio do espírito, para seu prazer pessoal e para o proveito da comunidade à qual pertencerá o seu trabalho futuro.

Albert Einstein

Conteúdo

Lista de Tabelas

Capítulo 4

Capítulo 5

Lista de Figuras

Capítulo 2

Capítulo 3

Capítulo 4

Capítulo 5

Capítulo 6

Capítulo 1

Introdução

O uso de especificações formais no processo de desenvolvimento de sistemas tem se tornado cada vez mais indispensável, principalmente no desenvolvimento de sistemas distribuídos e de sistemas que envolvem aplicações críticas. Em particular, tais sistemas englobam sistemas reativos e sistemas de tempo real, onde uma falha de operação pode causar enormes prejuízos ou danos irreparáveis.

Especificações formais são construídas usando-se uma linguagem formal ou um modelo matemático. Em conseqüência, este tipo de especificação possui características bastante interessantes para utilização no desenvolvimento de sistemas complexos. Entre outras vantagens, permitem a eliminação de inconsistências e ambigüidades que, potencialmente, são encontradas no projeto de tais sistemas [8]. Por exemplo, com o uso de especificações formais, o número de falhas na fase de implantação e implementação dos projetos é sensivelmente diminuído, uma vez que o número de erros gerados na sua fase de especificação tende a diminuir drasticamente.

Certas aplicações críticas apresentam características de sistemas distribuídos híbridos. Tais sistemas, de um modo geral, são sistemas resultantes da interconexão de sistemas de dinâmica contínua com sistemas de dinâmica discreta. Ou seja, são sistemas que apresentam características de dinâmica contínua regulada pela intervenção de eventos discretos [11, 10, 9, 12, 13]. Esta interdependência introduz maiores complicações e sutilezas no projeto desses sistemas, reforçando a necessidade de uma verificação rigorosa de suas propriedades.

Uma das formas de se especificar tais sistemas, com o rigor necessário, é através da utilização de autômatos híbridos [3]. Autômatos híbridos, resumidamente, são autômatos finitos cujos estados determinam o comportamento dinâmico do sistema, e cujas transições entre estados provocam uma alteração no seu perfil dinâmico. Diferentes componentes do sistema distribuído são modelados por autômatos independentes. O comportamento global do sistema é obtido da cooperação entre esses autômatos independentes. A comunicação entre os vários componentes do sistema é regulada por mensagens trocadas entre os autômatos par-

ticipantes, e também por uma memória compartilhada, à qual estes autômatos têm acesso. Normalmente, teorias clássicas de controle de sistemas discretos lidam, se tanto, com poucos componentes distribuídos cujas características dinâmicas são complexas. Sistemas modelados usando a noção de autômatos híbridos, por outro lado, apresentam vários componentes distribuídos cujas características dinâmicas são mais simples, porém determinantes.

Nesse trabalho, autômatos híbridos são usados para modelar alguns sistemas híbridos distribuídos reais. Num primeiro cenário, regiões de uma malha metroviária são modeladas formalmente. Numa segunda abordagem, são especificadas e modeladas partes de um sistema de gerenciamento de tráfego aéreo.

Na sua totalidade, um sistema metroviário é bastante complexo. Embora de topologia relativamente simples, esse sistema apresenta muitos subcomponentes e sensores, espalhados por toda a extensão da malha metroviária. Além disso, há intensa troca de mensagens e uma complexa atividade de coordenação entre esses componentes físicos. Para contornar essa complexidade, o sistema global foi simplificado e dividido em segmentos mais simples, os quais, então, foram modelados. Os segmentos selecionados enfocam propriedades distintas da operação da malha, e a conjunção dos resultados da análise dos vários segmentos contribui para a validação e a verificação do sistema total.

Um sistema de gerenciamento de tráfego aéreo coordena um conjunto de aviões autônomos, os quais competem por rotas em um mesmo espaço aéreo. Um subsistema de prevenção de colisão e alerta de tráfego, ou *Traffic alert and Collision Avoidance System* (TCAS), é usado para ajudar a gerenciar o tráfego aéreo como um todo. O TCAS é um protocolo para detecção e resolução de conflitos de rota entre aviões que operam em um mesmo setor de espaço aéreo. Sua tarefa é monitorar o tráfego aéreo nas proximidades do avião, detectar possíveis riscos de colisão e sugerir como resolver estes conflitos. Os modelos explorados neste trabalho consideram dois aviões voando no mesmo espaço aéreo, em direções opostas. Os conflitos surgidos durante a operação do sistema são analisados de modo a atestar sua segurança.

Na maioria das vezes, modelos construídos para sistemas híbridos realistas resultam num autômato de tal complexidade que se torna imprescindível a cooperação de ferramentas computacionais automáticas para análise da especificação produzida. Nesse trabalho, para analisar os modelos aqui desenvolvidos, foi utilizada a ferramenta HyTech [24], que é um verificador (semi) automático para autômatos híbridos lineares.

A dissertação é composta de sete capítulos. O primeiro capítulo é formado por esta breve introdução.

O Capítulo 2 discute a noção de autômatos híbridos. Neste capítulo, são apresentadas a definição formal da noção de autômato híbrido, suas características, exemplos de aplicação, e uma ferramenta computacional de suporte para a análise de modelos construídos com base na noção de autômatos híbridos.

O Capítulo 3 descreve os sistemas híbridos estudados neste trabalho: um segmento de malha metroviária e partes de um sistema de gerenciamento de tráfego aéreo. Ambos os sistemas apresentam características híbridas.

No Capítulo 4, é apresentada a verificação de propriedades da operação de uma região de malha metroviária. O núcleo deste capítulo é formado pelo artigo intitulado "Análise e Verificação de Segmentos de Via de uma Malha Metroviária", apresentado no *2nd Workshop on Formal Methods* (WFM'99), realizado em Florianópolis, Santa Catarina, em outubro de 1999.

O Capítulo 5 descreve a síntese de alguns parâmetros críticos para a operação segura de parte de uma malha metroviária. Seu conteúdo é formado pelo artigo "Formal Parameters Synthesis for Track Segments of a Subway Mesh", aceito para publicação na *7th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems* (ECBS'00), a ser realizada em abril de 2000, em Edinburgh, na Escócia. O artigo será publicado nos anais da conferência, sob a responsabilidade do *Institute of Electrical and Electronics Engineers* (IEEE).

O Capítulo 6 aborda partes de um sistema de gerenciamento de tráfego aéreo. Ali, é apresentada uma síntese de alguns parâmetros importantes para a operação segura dos aviões que trafegam num mesmo espaço aéreo. Esse capítulo é composto pelo artigo intitulado "Formal Verification and Synthesis for an Air Traffic Management System", submetido para a *3th International Conference on Formal Methods in Computer Aided Design* (FM-CAD'00), a ser realizada em novembro de 2000, em Austin, no Texas.

No Capítulo 7, são apresentadas algumas considerações e conclusões finais. São também oferecidas algumas sugestões para trabalhos futuros.

Capítulo 2

Autômatos Híbridos

Sistemas híbridos são sistemas cujo comportamento apresenta uma combinação de características discretas e contínuas. Os autômatos híbridos vêm se tornando um dos principais formalismos utilizados para a especificação desses sistemas. Basicamente, autômatos híbridos são autômatos finitos cujos estados descrevem o comportamento dinâmico do sistema modelado. Transições entre estados caracterizam uma mudança de perfil dinâmico e são sinalizadas por eventos, ou mensagens, recebidos de outros componentes ou do ambiente externo. Os vários componentes que fazem parte do sistema são modelados por autômatos independentes, sendo o comportamento do sistema como um todo obtido do produto dos autômatos componentes.

As duas primeiras seções deste capítulo apresentam dois exemplos típicos. As demais seções descrevem o formalismo de autômatos híbridos em maiores detalhes.

2.1 Um Exemplo Introdutório

Um exemplo bastante simples [27] é mostrado na Figura 2.1. O sistema modelado é o controlador de temperatura de um aquecedor. O modelo é descrito pela evolução determinística da variável x (temperatura), ao longo do tempo. O modelo é constituído por

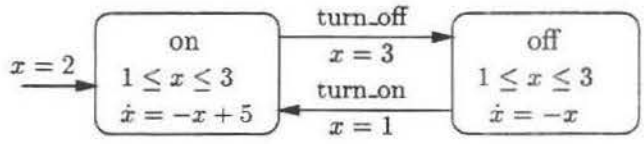


Figura 2.1: Autômato do Aquecedor

dois modos de operação, o modo ligado, cujo rótulo é *on*, e o modo desligado, cujo rótulo

é *off*. Cada modo de operação descreve um certo perfil dinâmico do sistema, através de uma equação diferencial. No modo *on*, o perfil dinâmico do sistema é descrito pela equação $dx/dt = -x+5$. Ou seja, quando o aquecedor está ligado a temperatura aumenta de acordo com a equação diferencial[1] $dx/dt = -x + 5$. No modo *off*, o perfil dinâmico do sistema muda para $dx/dt = -x$, indicando uma queda exponencial na temperatura do aquecedor. Na transição de *on* para *off* a condição de habilitação é $x = 3$. Estando o sistema no modo *on*, essa condição indica que, ao atingir 3 graus de temperatura, o aquecedor deve ser desligado, passando para o modo *off*, quando a temperatura passa a obedecer o novo perfil dinâmico $dx/dt = -x$. Nessa transição, a mensagem emitida é *turn_off*. De forma similar, o aquecedor permanece desligado, no modo *off*, até que a temperatura atinja 1 grau. Neste instante, o aquecedor é ligado, transitando para o modo *on* e emitindo a mensagem *turn_on*.

Para forçar uma mudança de perfil dinâmico, atribui-se à cada modo de operação uma condição invariante. O sistema só pode se manter num certo modo de operação enquanto a condição invariante daquele modo for satisfeita. Uma mudança de modo de operação pode, também, atribuir novos valores às variáveis do modelo. No exemplo, a condição invariante para ambos os modos de operação é $1 \leq x \leq 3$, ou seja, a mudança de modo deve ocorrer antes da temperatura deixar o intervalo de operação de [1,3] graus. Na verdade, como já citado, o aquecedor é desligado quando sua temperatura alcança exatamente 3 graus. E o aquecedor só é ligado quando a temperatura cai para exatos 1 grau. No caso ilustrado no exemplo, assume-se que o valor da variável x não é alterado quando as transições entre modos são acionadas.

A cada modo de operação está também associada uma condição inicial. A condição inicial no modo *on* é $x = 2$ graus. No modo *off*, a condição inicial é *false* e não está ilustrada na Figura 2.1. A evolução do sistema modelado é descrita a partir da condição inicial, das equações diferenciais presentes nos modos, e é influenciada também pelas transições entre modos.

2.2 Outro Exemplo

Um outro exemplo interessante [1] é apresentado na Figura 2.2. O cenário representa um jogo entre duas pessoas, um fugitivo e um perseguidor. Ambos trafegam numa pista circular de 40 metros. O perseguidor pode trafegar a uma velocidade de até 6 metros por segundo no sentido horário, ou pode também trafegar a 1/2 metro por segundo no sentido anti-horário. O fugitivo trafega a 5 metros por segundo, em ambas as direções. O perseguidor pode mudar o sentido de sua movimentação instantaneamente e a qualquer momento. No entanto, o fugitivo somente pode mudar sua direção de movimento em certos instantes de

[1] A notação $\dot{x}$ pode ser também usada em substituição à dx/dt.

tempo, separados por exatamente 2 segundos. O objetivo do fugitivo é evitar o perseguidor,

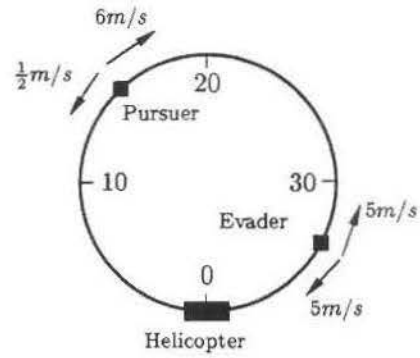


Figura 2.2: Cenário do Jogo da Perseguição

alcançando um helicóptero estacionado no marco zero da pista. O jogo é modelado pelo autômato híbrido mostrado na Figura 2.3. A variável p modela a posição do perseguidor, em relação à posição 0. Igualmente, a variável e modela a posição do fugitivo. O relógio t mede o tempo entre duas escolhas consecutivas de mudança de direção pelo fugitivo. Existem três modos de operação, segundo a direção de movimento e a posição do fugitivo: no sentido horário (*clockwise*), no sentido anti-horário (*counter*) e resgatado (*rescued*). A velocidade do fugitivo evolui de acordo com as equação $\dot{e} = 5$, quando este se desloca no sentido horário, e obedece a relação $\dot{e} = -5$, quando este se move no sentido anti-horário. A velocidade do perseguidor é ditada, de forma não-determinística, por um valor no intervalo $[-1/2, 6]$. Este não-determinismo modela a instantaneidade das decisões do perseguidor. A condição invariante nos modos *clockwise* e *counter* é $0 \leq e, p \leq 40$ refletindo a extensão da pista, de 40 metros.

É comum usar-se "guardas" [18] para representar as condições de habilitação para transições. Por exemplo, o guarda $p = 40 \rightarrow p := 0$ significa que a condição de habilitação é dada pelo predicado $p = 40$, e que a variável p, após a transição, assumirá o valor 0. Com esta interpretação em mente, a circularidade da pista é capturada, na Figura 2.3, pelos dois pares de transições superiores que têm origem e término nos mesmos modos de operação, *clockwise* e *counter*. A estratégia do fugitivo é capturada pelas outras quatro transições entre estes mesmos modos de operação. Estas transições implementam uma estratégia simples. O fugitivo sempre compara o tempo necessário para que ambos atinjam o helicóptero, assumindo movimento na direção horária. Se o seu tempo resultar inferior, o fugitivo permanece se deslocando no sentido horário. Caso contrário, o fugitivo inverte sua direção de movimento.

O modo *rescued* sinaliza que o fugitivo alcança o helicóptero. Portanto, a transição

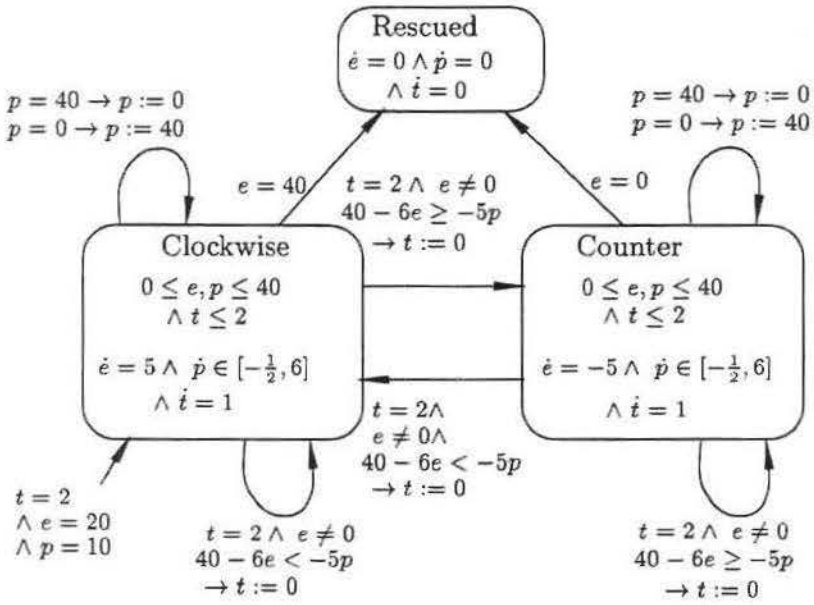


Figura 2.3: Autômato do Jogo da Perseguição

para este modo se dá tão logo o fugitivo alcance as marcas de 40 metros ou de 0 metros, quando se move no sentido horário ou no sentido anti-horário, respectivamente. Uma vez alcançado o helicóptero, as velocidades do fugitivo e do perseguidor, bem como a razão de variação do relógio, são zeradas, estancando a dinâmica do jogo.

Inicialmente, o fugitivo está na marca de 20 metros, diretamente oposto ao helicóptero, e o perseguidor está na posição de 10 metros, atrás do fugitivo. Partindo desta situação inicial, a evolução do autômato híbrido revelaria que o fugitivo alcançaria o helicóptero antes que o perseguidor o alcançasse. Portanto, a estratégia do fugitivo é eficaz, neste caso.

2.3 Definições

Um *autômato híbrido* é um sistema $A = (X, V, flow, init, inv, E, jump, \Sigma, syn)$, constituído pelos seguintes componentes:

Variáveis: Um conjunto finito $X = \{x_1, x_2, \ldots, x_n\}$ de variáveis. No caso do autômato híbrido que modela o aquecedor, apresentado na Figura 2.1, têm-se $n = 1$ e $X = \{x\}$.

Modos de Operação: Um conjunto finito V de modos de operação. No exemplo do aquecedor, $V = \{on, off\}$.

Condições de Atividade Contínua: É dada pelo componente *flow*. Para todo modo $v \in V$, *flow(v)* é um predicado sobre o conjunto de variáveis $X \cup \dot{X}$, onde $\dot{X} = \{\dot{x}_1, \ldots, \dot{x}_n\}$. Enquanto o modo de operação de A é v, as variáveis em X evoluem de acordo com uma curva diferencial onde os valores das variáveis reais e suas derivadas primeira satisfazem a condição de atividade contínua *flow(v)*. No exemplo da Figura 2.1, o modo de operação *on* tem uma condição de atividade contínua dada pelo predicado $\dot{x} = -x + 5$, e no modo *off* essa condição é dada por $\dot{x} = -x$. A especificação de uma atividade contínua pode envolver não-determinismo. Por exemplo, se *flow(v)* for da forma $\dot{x} \in [2, 4]$, esta atividade contínua, não-determinística, representaria o predicado $2 \leq \dot{x} \leq 4$. Isso significa que o valor de $\dot{x}$ seria fixado, não-deterministicamente, em qualquer valor no intervalo $[2, 4]$.

Condições Invariantes: É dada pelo componente *inv*. Para todo modo $v \in V$, *inv(v)* é um predicado sobre as variáveis em X. Enquanto o modo de operação do autômato é v, as variáveis em X devem satisfazer à condição invariante *inv(v)*. No exemplo da Figura 2.1, em ambos os modos de operação a condição invariante é $1 \leq x \leq 3$.

Condições Iniciais: É dada pelo componente *init*. Para todo modo $v \in V$, *init(v)* é um predicado sobre as variáveis em X. O autômato A pode iniciar no modo v quando a condição inicial *init(v)* é verdadeira. Na representação gráfica, as condições iniciais aparecem como rótulos em arcos de entrada sem pontos de origem. As condições falsas não são ilustradas. No modo *on*, no exemplo da Figura 2.1, a condição inicial é $x = 2$, e no modo *off* a condição inicial é *false*.

Chaves de Controle ou Transições: É formado por um multi-conjunto E de arestas (v, v'), onde $v, v' \in V$. Na Figura 2.1, existem duas transições, (*on*, *off*) e (*off*, *on*). Ou seja, existem duas arestas, uma de *on* para *off* e outra de *off* para *on*.

Condições de Mudança de Fase: É dada pelo componente *jump*. Para toda transição $e \in E$, *jump(e)* é um predicado sobre as variáveis em $X \cup X'$, onde $X' = \{x'_1, \ldots, x'_n\}$. O símbolo primitivo x_i, para $1 \leq i \leq n$, refere-se ao valor da variável x_i antes da mudança do modo de operação, e o símbolo derivado x'_i refere-se ao valor atribuído à variável x_i após a mudança do modo de operação. Desta forma, a condição *jump* relaciona o valor das variáveis reais antes da mudança do modo de operação com os possíveis valores dessas variáveis após a mudança do modo de operação. No exemplo do aquecedor, a transição (*on*, *off*) tem uma condição de mudança de fase dada pelo predicado $x = 3 \wedge x' = x$, e a transição (*off*, *on*) tem a condição de mudança de fase dada pelo predicado $x = 1 \wedge x' = x$. Nas ilustrações, condições triviais na forma $x' = x$ não são indicadas. No caso do exemplo, portanto, o valor das variáveis reais não muda quando há uma transição entre modos.

Eventos ou Mensagens de Sincronização: É dado por um conjunto Σ de eventos, junto com uma função, *syn*, que associa um evento de Σ a cada transição $e \in E$. No exemplo, tem-se $\Sigma = \{turn\_on, turn\_off\}$. À transição (on, off) corresponde o evento *turn_off* e à transição (off, on) corresponde o evento *turn_on*. Os eventos permitem a sincronização entre autômatos híbridos distribuídos, como será visto ainda neste capítulo.

De ora em diante, exceto se houver menção explícita em contrário, um autômato híbrido A denotará sempre um sistema na forma $(X, V, flow, init, inv, E, jump, \Sigma, syn)$. Uma referência a um autômato híbrido A_i denotará um sistema semelhante, cujos componentes são referenciados pelo mesmo índice i.

2.4 O Autômato Produto

Um sistema distribuído, normalmente, é formado por vários componentes operando concorrentemente e comunicando-se uns com os outros. O sincronismo do sistema global é capturado de duas maneiras: (i) usando variáveis compartilhadas; ou (ii) forçando com que transições ocorram no mesmo evento. O uso de variáveis compartilhadas dispensa explicações. Para se obter sincronismo através do segundo mecanismo, calcula-se o autômato produto, descrito a seguir, tendo como base todos os autômatos individuais que compõem o modelo do sistema. Àquelas transições, nos autômatos participantes, as quais está associado um mesmo evento, corresponderá uma única transição, simultânea, no autômato produto.

Se A_1 e A_2 são autômatos híbridos, então o *autômato produto* de A_1 e A_2 é um sistema

$$A = (X_1 \cup X_2, V_1 \times V_2, flow, init, inv, E, jump, \Sigma_1 \cup \Sigma_2, syn)$$

onde
$$\begin{aligned} flow((v_1, v_2)) &= flow(v_1) \wedge flow(v_2), \\ inv((v_1, v_2)) &= inv(v_1) \wedge inv(v_2), \\ init((v_1, v_2)) &= init(v_1) \wedge init(v_2). \end{aligned}$$

E, ainda, $e = ((v_1, v_1'), (v_2, v_2')) \in E$ se e somente se uma das três condições abaixo é verificada:

1. $v_1 = v_1'$, $e_2 = (v_2, v_2') \in E_2$, $syn_2(e_2) \notin \Sigma_1$.
   Daí, $jump(e) = jump_2(e_2)$ e $syn(e) = syn_2(e_2)$; ou

2. $v_2 = v_2'$, $e_1 = (v_1, v_1') \in E_1$ e $syn_1(e_1) \notin \Sigma_2$.
   Daí, $jump(e) = jump_1(e_1)$ e $syn(e) = syn_1(e_1)$; ou

3. $e_1 = (v_1, v_1') \in E_1$, $e_2 = (v_2, v_2') \in E_2$, $syn_1(e_1) = syn_2(e_2)$.
   Daí, $jump(e) = jump_1(e_1) \wedge jump_2(e_2)$ e $syn(e) = syn_1(e_1)$.

Portanto, as transições e_1 e e_2 são executadas simultaneamente no autômato produto quando $syn_1(e_1) = syn_2(e_2)$, isto é, quando a ambas está associado um mesmo evento. Por outro lado, essas transições são assíncronas no autômato produto se o evento $syn_1(e_1)$ não ocorre no conjunto de eventos de A_2, e nem $syn_2(e_2)$ ocorre no conjunto de eventos de A_1.

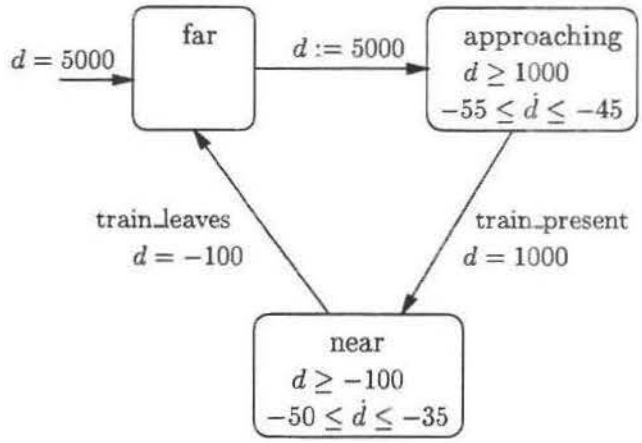


Figura 2.4: Autômato do trem na ferrovia

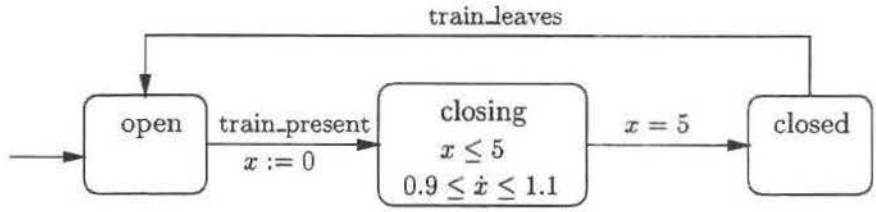


Figura 2.5: Autômato do controle da cancela

Como exemplo, imagine uma ferrovia com uma passagem de nível onde foi instalada uma cancela. Um trem trafega pela via e se aproxima da passagem de nível. Deseja-se verificar se o trem passa em segurança pelo trecho da cancela. Ou seja, a cancela deve estar fechada quando o trem atravessar esse trecho da via. Inicialmente, o trem se aproxima da passagem de nível com uma velocidade qualquer no intervalo de $[-55, -45]$ metros por unidade de tempo. Um sensor está localizado a uma distância de 1000 metros da cancela. Em um certo momento, o trem passa pelo sensor que, então, comanda o início do fechamento da cancela. A cancela começa a se fechar, devendo concluir o movimento em no máximo 5 unidades de tempo. No entanto, o mecanismo que opera a cancela introduz uma imprecisão de até

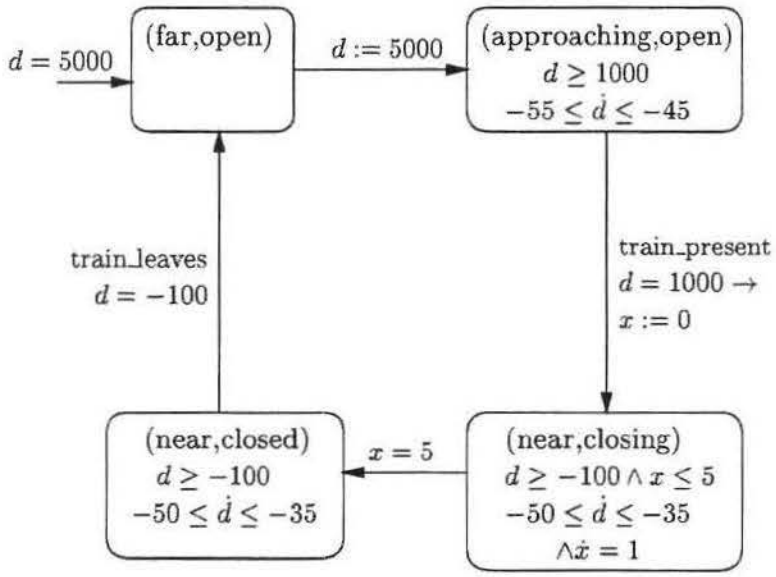


Figura 2.6: Autômato produto do trem com o controlador

10%, para mais ou para menos, no tempo de movimentação da cancela. Assim, a cancela pode completar a operação de fechamento num ponto qualquer entre 4.5 e 5.5 unidades de tempo, após iniciar seu movimento. Quanto ao trem, após a passagem pelo sensor, este diminui sua velocidade para um valor arbitrário no intervalo $[-50, -35]$ metros por unidade de tempo, e mantém esta velocidade até cruzar e ultrapassar a região da cancela. À uma distância de 100 metros, do outro lado da cancela, o trem passa por um segundo sensor. Este segundo sensor comanda o início do movimento para abrir a cancela novamente.

A Figura 2.4 [26] ilustra o autômato do trem e a Figura 2.5 mostra o autômato que modela o controle de movimentação da cancela. O autômato produto pode ser visualizado na Figura 2.6. Observe como eventos com o mesmo rótulo no autômato do trem e no autômato do controlador forçam o sincronismo no autômato produto. O evento *train_present* sinaliza a passagem do trem pelo primeiro sensor, e o evento *train_leaves* sincroniza os autômatos quando o trem sai da região de perigo. Isso garante que o autômato da cancela e o autômato do trem sincronizam suas transições quando o trem passa pelos sensores. Na Figura 2.6, foram descartados alguns modos do autômato produto. Por exemplo, o modo $(far, closed)$ foi descartado. Estes modos representam situações impossíveis de ocorrer, ou seja, estes modos não são alcançáveis, conforme é explicado na Seção 2.5.

A ferramenta HyTech (veja a Seção 2.12) é capaz de construir o produto de autômatos híbridos automaticamente.

2.5 Configurações e Trajetórias

Uma *configuração* de um autômato híbrido A é um par (v, a) consistindo de um modo de operação $v \in V$ e de um vetor de valores reais $a = (a_1, \ldots, a_n)$. Esse último contém um valor $a_i \in \mathbb{R}$ para cada variável $x_i \in X$. A configuração (v, a) é *admissível* se o predicado *inv(v)* é verdadeiro quando a variável x_i é substituída pelo valor a_i, para $i = 1, \ldots, n$. Observe, no exemplo da Figura 2.1, que a configuração $(on, 1.5)$ é admissível. Já a configuração $(on, 0.5)$ não é admissível. A configuração (v, a) é *inicial* se o predicado *init(v)* for verdadeiro quando todo x_i for substituído pelo correspondente valor a_i. No mesmo exemplo, é fácil ver que a configuração $(on, 2)$ é inicial.

Considere um par (q, q') formado por duas configurações admissíveis, $q = (v, a)$ e $q' = (v', a')$. O par (q, q') é uma *mudança de fase* de q para q' em A se existe uma transição $e \in E$, com origem no modo v e destino no modo v', e tal que o predicado *jump(e)* seja verdadeiro quando toda variável x_i é substituída pelo valor a_i, e toda variável x_i' é substituída pelo valor a_i'. No exemplo da Figura 2.1, existem duas mudanças de fase, ((*on*,3),(*off*,3)) e ((*off*,1),(*on*,1)). Quando ocorre uma mudança de fase de q para q', diz-se que v' é um modo de operação sucessor de v.

Dadas duas configurações admissíveis de A, $q = (v, a)$ e $q' = (v', a')$, o par (q, q') é uma *atividade contínua* de A se $v = v'$, se existe um real não negativo $\delta \in \mathbb{R}$ (a duração da atividade contínua) e se existe também uma função diferenciável $\rho : [0, \delta] \to \mathbb{R}^n$ (a curva da atividade contínua), tais que os três requisitos que se seguem sejam satisfeitos:

1. Pontos inicial e final: $\rho(0) = a$ e $\rho(\delta) = a'$.
2. Condição invariante: Para todo instante de tempo $t \in [0, \delta]$, a configuração $(v, \rho(t))$ é admissível.
3. Condição de atividade contínua: Tome $\dot{\rho} : [0, \delta] \to \mathbb{R}^n$ como sendo a derivada primeira de ρ. Para todo instante de tempo $t \in [0, \delta]$, o predicado *flow(v)* é verdadeiro quando cada variável x_i é substituída pela i-ésima coordenada do vetor $\rho(t)$, e toda variável $\dot{x}_i$ é substituída pela i-ésima coordenada do vetor $\dot{\rho}(t)$.

No exemplo da Figura 2.1, os pares $((off, 3), (off, 2))$ e $((off, 3), (off, 2.5))$ são atividades contínuas. Quando (q, q') é uma atividade contínua, a' é dito um ponto sucessor de a. Note que um ponto é sucessor dele mesmo, visto que sempre existe a atividade contínua com duração 0 (zero).

Uma *trajetória* de um autômato híbrido A é uma seqüência finita $q_0, q_1, \ldots, q_k$, de configurações admissíveis onde q_0 é uma configuração inicial de A, e onde todo par (q_j, q_{j+1}) de configurações consecutivas da seqüência é uma mudança de fase ou uma atividade contínua de A. Uma configuração q de A é *alcançável* se q é a última configuração de alguma trajetória de A. No exemplo da Figura 2.1, todas as configurações admissíveis são alcançáveis.

2.6 Computação de Configurações Alcançáveis

Uma *região convexa* de dimensão n é um poliedro convexo em $\mathbb{R}^n$. Uma *região* é uma união finita de regiões convexas. Dado um predicado φ, a região determinada por φ é denotada por $[\varphi]$, e é chamada de região φ.

Dada uma região φ, $Post(\varphi)$ designa a *região alcançável* a partir da região φ. Em geral, $Post(\varphi)$ reúne todas aquelas configurações q' para as quais existe uma configuração q em φ tal que a configuração q' é alcançável a partir de q através de uma trajetória de A constituída de uma atividade contínua, seguida, se possível, de uma mudança de fase. Partindo de $\varphi_0 = \varphi$, a iteração desse processo computará as regiões $\varphi_{k+1} = Post(\varphi_k)$, para $k = 1, 2, \ldots$. Se atingirmos uma região φ_k tal que $\varphi_k = \varphi_{k+1}$, então φ_k caracteriza todas as configurações alcançáveis por trajetórias de A partindo de φ_0.

Para se aplicar esse processo é preciso, primeiramente, calcular a região $Post(\varphi)$ de maneira eficiente, dada a região φ. Também é necessário que a computação convirja após um número finito de iterações. É possível satisfazer a primeira restrição considerando-se uma subclasse restrita dos autômatos híbridos, os autômatos híbridos lineares [27, 29, 23], apresentados na Seção 2.7. A segunda restrição pode ser satisfeita tomando-se uma subclasse dos autômatos híbridos lineares, os *timed-automata*, onde as variáveis são todas relógios, isto é, são variáveis cuja derivada primeira é sempre 1. A ferramenta HyTech (veja Seção 2.12) trabalha com autômatos híbridos lineares, englobando a primeira restrição. A segunda restrição não é contemplada pela ferramenta. Isto é, pode não haver convergência no processo iterativo. Quando o processo converge, a região φ_k é denotada por $Post^*(\varphi_0)$.

Existe uma outra forma de se calcular regiões alcançáveis. Dada uma região φ, a região $Pre(\varphi)$ contém uma configuração q se existe uma configuração q' em φ tal que q' é alcançável a partir de q através de uma trajetória de A formada por uma atividade contínua, seguida, se possível, de uma mudança de fase. Iniciando com uma região ψ_0 e iterando-se esse processo, pode-se obter as regiões ψ_{k+1}, onde $\psi_{k+1} = Pre(\psi_k)$. Se o processo atingir uma região ψ_k tal que $\psi_{k+1} = \psi_k$, então ψ_k representa a região a partir da qual pode-se alcançar a região original, ψ_0. Neste caso, a região ψ_k é denotada por $Pre^*(\psi_0)$.

2.7 Autômatos Híbridos Lineares

A não-linearidade das características dinâmicas de um sistema real pode ser um fator complicante para a análise do modelo. Nessas situações, são usados métodos de linearização que transformam o modelo original do sistema num autômato híbrido linear [23, 27], mais conveniente para análise, e usualmente não-determinístico. A necessidade de linearização também é imposta pelo fato de que as ferramentas (semi) automáticas disponíveis para a análise das especificações lidam, via de regra, apenas com autômatos híbridos lineares [24].

Um *predicado atômico linear* é uma inequação entre constantes racionais e combinações lineares de variáveis com coeficientes racionais, como por exemplo $2x + 4y - 7z/2 \leq -10$. Um *predicado linear convexo* é uma conjunção finita de predicados atômicos lineares, e um *predicado linear* é uma disjunção finita de predicados lineares convexos. Um autômato híbrido A é um autômato híbrido linear, se satisfaz os seguintes requisitos:

1. **Linearidade**: Para todo modo de operação $v \in V$, a condição de atividade contínua *flow(v)*, a condição invariante *inv(v)* e a condição inicial *init(v)*, são predicados lineares convexos. E também, para toda transição $e \in E$, a condição de mudança de fase *jump(e)* é um predicado linear convexo;

2. **Atividade contínua independente**: Para todo modo de operação $v \in V$, a condição de atividade contínua *flow(v)* é um predicado sobre as variáveis que estão no conjunto $\dot{X}$ somente, não contendo variáveis do conjunto X.

É um tanto limitante para a modelagem proibir atividades contínuas, tais como $\dot{x} = x$. Por outro lado, é permitido o uso de variáveis tais como relógios, cronômetros e relógios com derivas. É possível, por exemplo, especificar $\dot{x} \in [1-\epsilon, 1+\epsilon]$, para alguma constante ϵ. Nesse caso $\dot{x} = 1 - \epsilon$ representa um relógio com atraso ϵ, e $\dot{x} = 1 + \epsilon$ representa um relógio com avanço ϵ. Nesses dois casos a condição de linearidade é atendida.

Como um exemplo, observe o autômato ilustrado na Figura 2.7 ignorando, por ora, a presença das variáveis y e z. Este autômato resulta de uma linearização simples do autômato original, mostrado na Figura 2.1. No autômato da Figura 2.1, no modo *on*, o valor de x está restrito ao intervalo $[1, 3]$. Como, nesse modo, o valor de $\dot{x}$ é dado por $\dot{x} = -x + 5$, conclui-se que, nesse mesmo modo, o valor de $\dot{x}$ está restrito ao intervalo $[2, 4]$. Este é o intervalo que a especificação atribui à variável $\dot{x}$, para o modo *on*, no autômato ilustrado na Figura 2.7. De forma similar, no modo *off*, a função $-x$ é substituída pelo intervalo $[-3, -1]$. Neste processo de linearização, cada atividade contínua não-linear é relaxada para um intervalo de valores que compreende a solução da equação diferencial original.

2.8 Linearização de Autômatos Híbridos

O autômato do aquecedor ilustrado na Figura 2.1 não é um autômato híbrido linear, já que a condição de atividade contínua *flow(v)* não é um predicado sobre o conjunto $\dot{X}$ somente. Existem duas técnicas para substituir um autômato híbrido não-linear por um autômato híbrido linear [27]. A primeira técnica, chamada *clock translation*, substitui as variáveis que causam a não-linearidade por relógios. A segunda técnica, chamada *linear phase-portrait approximation*, substitui predicados não-lineares por predicados lineares mais relaxados.

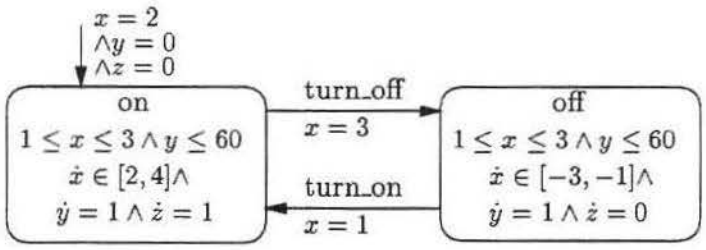


Figura 2.7: Autômato Aquecedor com verificação de segurança

A idéia da transformação tipo *clock translation* é que, muitas vezes, o valor de uma variável x pode ser determinado a partir: (i) do tempo desde a última reatribuição de x para uma constante; e (ii) do valor desta constante. Neste caso, as condições invariante, inicial, e de mudança de fase sobre a variável x podem ser traduzidas para condições sobre um relógio t^x. O relógio t^x é reiniciado a todo instante em que x é reatribuído para uma nova constante. Se necessário, os modos de operação podem ser duplicados para computar reatribuições de x para diferentes constantes, refinando mais o modelo.

Formalmente, a variável x de um autômato híbrido A é dita *clock-translatable* se os dois requisitos a seguir são verificados:

1. Condição de solução: em cada condição de atividade contínua *flow(v)*, todas as ocorrências de x e $\dot{x}$ estão na forma $\dot{x} = g^v(x)$, onde $g^v : \mathbb{R} \to \mathbb{R}$ é uma função integrável e com sinal constante sobre o intervalo da condição invariante. Nas condições invariante, inicial, e de mudança de fase, todas as ocorrências de x e x' são da forma $x' = x$, ou $x \sim c$, ou $x' \sim c$, onde $\sim$ é um operador de desigualdade, e c é uma constante racional.

2. Condição de inicialização: para todo modo de operação v, a condição inicial *init(v)* é da forma $x = c$ para alguma constante c. E para toda transição (v, v'), é necessário que: (i) $g^v = g^{v'}$; e (ii) $jump(v, v')$ seja da forma $x' = x$, isto é, seu valor não muda, ou que $jump(v, v')$ seja da forma $x' = c$, para alguma constante c.

No *linear phase-portrait approximation* a idéia é relaxar os predicados *flow*, *invariant*, *initial*, e *jump* de modo que se obtenham condições lineares. Em seguida, cada predicado não-linear p é substituído por um predicado linear p' tal que p implica p'. O autômato híbrido linear da Figura 2.7 é um *linear phase-portrait approximation* do sistema aquecedor, ilustrado na Figura 2.1. As condições invariantes, iniciais, e de mudança de fase já são lineares no modelo original. Somente as condições de atividade contínua precisam ser relaxadas. Para o modo de operação *on*, a condição invariante $1 \leq x \leq 3$ e a condição

de atividade contínua $\dot{x} = -x + 5$ implicam que este predicado pode ser relaxado para condição linear $\dot{x} \in [2, 4]$. Do mesmo modo, sobre o modo de operação *off*, a condição de atividade contínua pode ser relaxada para a condição linear $\dot{x} \in [-3, -1]$.

Observe que a transformação *clock translation* preserva as trajetórias de um sistema, enquanto que a transformação *linear phase-portrait approximation* adiciona trajetórias ao sistema original. Dessa forma, se uma propriedade é satisfeita para o sistema relaxado, então a propriedade é satisfeita para o sistema original. Por outro lado, se o sistema relaxado viola uma propriedade de segurança, então deve ser verificado se a trajetória insegura é uma trajetória válida no sistema original. Se não, a análise é inconclusiva e as aproximações devem ser refinadas, como por exemplo, dividindo os modos de operação.

Pode-se mostrar o seguinte teorema [3]: se A é um autômato híbrido linear, e se φ é uma região de configurações para A, então o cálculo de $Post^*(\varphi)$ converge e o resultado será uma nova região de configurações para A. Esse teorema permite automatizar a análise dos autômatos híbridos lineares.

2.9 Requisitos de Segurança

Um requisito de segurança é um conjunto de restrições e condições impostas às configurações do sistema modelado. Uma configuração é dita segura se satisfaz à todas as condições de segurança que estão associadas a si. Caso contrário a configuração é dita insegura. Os requisitos de segurança são satisfeitos pelo sistema como um todo se e somente se todas as configurações alcançáveis são seguras. Ou seja, dentre todas as configurações alcançáveis não devem existir configurações inseguras. Geralmente, um requisito de segurança é especificado pela descrição de combinações de valores desejáveis e principalmente combinações de valores indesejáveis para as variáveis do sistema.

Para um autômato híbrido, os requisitos de segurança podem ser especificados através de asserções associadas aos modos de operação. Uma asserção φ sobre modos de operação é uma função que atribui a todo modo $v \in V$ um predicado $\varphi(v)$ sobre as variáveis em X. A asserção sobre um modo de operação é verdadeira ou falsa para uma configuração (v, a), se o predicado $\varphi(v)$ é verdadeiro ou falso quando toda variável x_i é substituída pelo valor correspondente a_i.

Um requisito de segurança também pode ser especificado criando-se um predicado *unsafe* sobre as configurações de A. Esse predicado descreve as condições que violam a segurança do sistema. Assim, o sistema modelado é seguro se a região *unsafe* de configurações de A não incluir nenhuma configuração alcançável de A. Para verificar se um autômato híbrido A satisfaz um requisito de segurança especificado por uma asserção *unsafe*, especificada sobre suas configurações, calcula-se uma outra asserção sobre configurações, *reach*, que caracteriza todas as configurações de A alcançáveis a partir de configurações iniciais. A região *reach* é

computada calculando-se a região $Post^*(\varphi_0)$, onde φ_0 é a região inicial de A. Em seguida, verifica-se se as regiões *reach* e *unsafe* têm um ponto em comum, ou seja, se a intersecção entre essas regiões não é vazia. Em caso afirmativo, o requisito de segurança é violado. Esse sendo o caso, a ferramenta HyTech é capaz de indicar uma trajetória de A que leva à região *unsafe*, a partir da qual pode-se identificar, no modelo, quais condições de operação levam à violação de segurança.

Para o autômato exemplo da Figura 2.1, um requisito de segurança poderia ser o seguinte: o aquecedor não deve ficar ligado mais de 2/3 do tempo nos primeiros 60 minutos de operação. Pode-se usar uma variável y para modelar o tempo total de operação do aquecedor e uma variável z para representar o tempo total gasto apenas enquanto o aquecedor está ligado. Observe que a introdução dessas variáveis auxiliares não altera o comportamento do autômato. No exemplo, a condição insegura *unsafe* seria especificada através do predicado $y = 60 \wedge z > 2y/3$, válido para os dois modos de operação. Na Figura 2.7 é apresentado o autômato modificado da Figura 2.1 com as variáveis auxiliares, y e z, acrescentadas ao modelo. A variável y é um *relógio* ($\dot{y} = 1$ para todos os modos de operação) e z é um *cronômetro* ($\dot{z} = 0$ ou $\dot{z} = 1$ em todo modo de operação). A cláusula $y \leq 60$ foi acrescentada aos dois modos de operação, *on* e *off*, para garantir que a simulação não ultrapassa os primeiros 60 minutos de operação. As demais condições ilustradas na Figura 2.7 advém da linearização do modo original, conforme já discutias na Seção 2.7. A relaxação resulta num autômato linear que, se demonstrado seguro, garante também a segurança do sistema original.

2.10 Verificação de Propriedades

De um modo geral, as propriedades a serem verificadas pelo modelo são expressas na forma de sentenças lógicas escritas numa linguagem formal apropriada, envolvendo componentes da especificação. A verificação da propriedade eqüivale à demonstração da veracidade da sentença descritiva.

No autômato da Figura 2.7, o objetivo é verificar que o aquecedor não permanece ligado mais que 2/3 do tempo na primeira hora de operação. A região de configurações especificada como *unsafe*, para ambos os modos de operação, é dada pelo predicado $y = 60 \wedge z \geq 2y/3$. A computação das configurações alcançáveis começa na região $\varphi_0 = \{(on, x = 2 \wedge y = 0 \wedge z = 0), (off, false)\}$. A configuração inicial é $(on, 2)$. Em seguida, calcula-se a região $\varphi_1 = Post(\varphi_0)$ em dois passos. Num primeiro passo, calcula-se a região alcançável através de atividades contínuas a partir da região φ_0. Num segundo passo, calcula-se a região alcançável através de uma mudança de fase a partir dessa região intermediária. O predicado obtido para φ_1 seria $(x \leq 3) \wedge (2z + 2 \leq x \leq 4z + 2) \wedge (y = z)$. A ferramenta HyTech realiza esta computação automaticamente até que as regiões não evoluam mais. Após 73 iterações,

a região *reach* é calculada e determina-se que as regiões *reach* e *unsafe* não têm pontos em comum. Isso garante que a propriedade desejada é verificada pelo autômato linear. Como as trajetórias deste autômato incluem todas as trajetórias do autômato híbrido não-linear original, da Figura 2.1, conclui-se que os requisitos de segurança são também satisfeitos pelo autômato original.

2.11 Síntese de Parâmetros e Análise Paramétrica

Parâmetros são constantes simbólicas que assumem valores fixos. Uma análise paramétrica produz intervalos de valores para as constantes simbólicas do modelo, e de forma que a segurança do sistema seja ainda mantida. Com isso, podem ser sintetizados valores máximos e valores mínimos para os parâmetros introduzidos no modelo.

Em um autômato híbrido, um parâmetro α pode ser representado por uma variável que nunca muda de valor, isto é, em todo modo de operação inclui-se a condição $\dot{\alpha} = 0$. Além disso, em todas as transições especifica-se a condição $\alpha' = \alpha$. Assim, o parâmetro α mantém o mesmo valor em todas as configurações ao longo de uma trajetória do autômato. Um valor $a \in \mathbb{R}$ é dito seguro para um parâmetro α se nenhuma configuração insegura se torna alcançável quando a restrição $\alpha = a$ é adicionada à todas as condições iniciais do modelo.

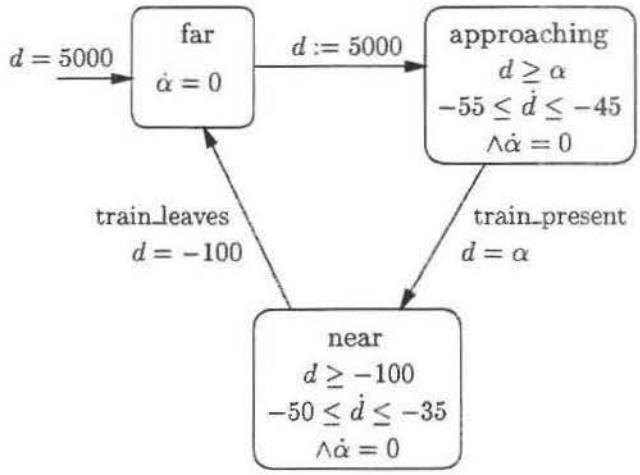


Figura 2.8: Autômato do trem parametrizado

Na Figura 2.8 o autômato do trem, cujo original foi apresentado na Figura 2.4, foi parametrizado substituindo-se a variável α pela constante 1000. Esta última constante indicava a que distância da cancela se encontrava o primeiro sensor. Portanto, no modelo

parametrizado, o controlador deve detectar a presença do trem quando esse se encontra a α metros da travessia da cancela. Usando essa nova descrição do modelo, a ferramenta HyTech informa que se deve impor $\alpha > 287\frac{7}{9}$ metros para que o requisito de segurança seja satisfeito. O primeiro sensor, portanto, deve estar localizado a mais de $287\frac{7}{9}$ metros da cancela para que o sistema opere de forma segura.

2.12 O Analisador HyTech

Em geral, o exercício de modelagem produz autômatos de uma complexidade tal que, para se obter uma análise do modelo, faz-se necessário o suporte de ferramentas computacionais automáticas. O software HyTech [25, 27, 26, 24] é uma ferramenta automática que pode ser usada na análise, verificação de propriedades e síntese de parâmetros para sistemas híbridos realistas. O HyTech é um verificador de autômatos híbridos lineares.

O arquivo de entrada da ferramenta possui duas partes. A primeira parte descreve os autômatos híbridos lineares participantes do modelo. O autômato produto é calculado automaticamente. A análise prossegue a partir desse autômato produto. A segunda parte do arquivo de entrada possui uma seqüência de comandos que direcionam a computação por parte da ferramenta. A linguagem de especificação desses comandos é muito semelhante a uma linguagem de programação, com tipos de dados primitivos, e com operações primitivas, tais como, *Post* e *Pre*, além de operadores binários e quantificação existencial. Para maior facilidade de uso, existem macros já construídas que auxiliam a análise de alcançabilidade, a análise paramétrica e a geração de trajetórias de erro.

A seguir, à título de ilustração, é apresentada a seqüência de comandos utilizada no sistema do aquecedor, onde α é um parâmetro que representa o tempo em que o aquecedor se mantém ligado na primeira hora de operação.

```
unsafe := y=60 & z >= alpha;                                    (1)
reachable :=                                                    (2)
        reach forward from init_states endreach;                (3)
bad_alpha_values := omit all locations                          (4)
        hide non_parameters in reachable & unsafe endhide;      (5)
good_alpha_values := ~bad_alpha_values;                         (6)
    prints ''Bad values:'';                                     (7)
        print omit all locations bad_alpha_values;              (8)
    prints ''Good values:'';                                    (9)
        print omit all locations good_alpha_values;             (10)
```

Na linha (1) é especificada a configuração insegura. Nas linhas (2) e (3) são computadas as configurações alcançáveis a partir das configurações iniciais, iterando-se o operador *Post*.

As linhas (4) e (5) especificam o predicado que caracteriza as configurações inseguras alcançáveis. O símbolo $\sim$, na linha (6), significa negação. As linhas de (7) a (10) produzem a saída. O resultado é:

```
Bad values:
      alpha <= 36
Good values:
      alpha > 36
```

Portanto, conclui-se que $\alpha > 36$ é suficiente para satisfazer os requisitos de segurança do aquecedor. Ou seja, o aquecedor nunca fica ligado mais de 36 minutos durante a primeira hora de operação. Esse resultado estabelece que o sistema é seguro.

Uma outra vantagem da ferramenta é a geração de trajetórias de erro quando o sistema falha para um determinado requisito de segurança. Geralmente, isto é usado para se depurar o modelo e se encontrar os erros de especificação e mal funcionamento do modelo, dados certos parâmetros.

A ferramenta HyTech pode verificar automaticamente que o modelo do aquecedor linearizado, ilustrado na Figura 2.7, satisfaz todos os requisitos de segurança propostos [27]. Da mesma forma, para o jogo ilustrado na Figura 2.2, a ferramenta verifica que a estratégia do fugitivo é eficaz, permitindo-lhe sempre alcançar o helicóptero antes de ser alcançado pelo perseguidor [1].

O HyTech, a princípio, foi usado para analisar sistemas de controle, como é o caso do exemplo do aquecedor e do trem. Hoje, é uma das únicas ferramentas (semi) automáticas de apoio à análise de modelos baseados em autômatos híbridos lineares, e tem sido usado em diversas situações [3, 27, 26, 24]. Existem outras ferramentas voltadas à análise de especificações de sistemas de tempo real, tal como o UPPAAL [33, 6, 7, 5]. Este último é um verificador baseado em *timed-automata*, uma subclasse mais restrita dos autômatos híbridos lineares, pois suporta apenas a noção de relógios, não permitindo equações diferenciais mais complexas para descrever perfis dinâmicos. Inúmeros outros trabalhos relacionados podem ser encontrados em [30, 28, 29].

Capítulo 3

Alguns Sistemas Híbridos Reais

Sistemas híbridos fazem parte de uma classe de sistemas englobados pelos sistemas reativos e de tempo real. Sistemas desta natureza, usualmente, realizam aplicações críticas. Devido as características híbridas dessas aplicações, esse tipo de sistema têm se tornado cada vez mais estudado. Isso tem ocorrido, principalmente, porque as aplicações são complexas e porque envolvem altos riscos de perdas e danos que seriam causados por eventuais falhas de operação. Entre essas aplicações, destacam-se os sistemas de controle de reatores nucleares e os sistemas de controle de tráfego aéreo ou metroviário. Neste capítulo são descritos dois sistemas que serão alvo de modelagem e análise neste trabalho: uma malha metroviária e um sistema de controle de tráfego aéreo.

3.1 Descrição do Segmento de Malha Metroviária

As malhas metroviárias, atualmente, apresentam uma grande complexidade devido a crescente demanda por transporte coletivo. A automação das malhas ocorreu em função da necessidade de se implantar melhorias e aumentar sua capacidade de transporte com níveis compatíveis de disponibilidade. Um exemplo do avanço tecnológico implantado neste tipo de sistema é o uso de circuitos microprocessados, os quais são vantajosos no que diz respeito a aspectos de controle e supervisão. Esta automação, por sua vez, demanda uma verificação cuidadosa dos aspectos de segurança do sistema. A ênfase da modelagem e análise, realizadas neste trabalho, está voltada para esses aspectos de segurança. O objetivo é, de um lado, modelar o sistema, validando seu funcionamento operacional e verificando algumas de suas propriedades. Por outro lado, deseja-se também sintetizar valores mais apropriados para alguns parâmetros críticos do sistema, de forma a se obter uma maior eficiência e uma melhor eficácia na sua operação.

Para descrever uma malha metroviária é preciso, antes de tudo, ressaltar dois pontos importantes. Em primeiro lugar, o sistema modelado apresenta o requisito básico de um

sistema híbrido, que é a mesclagem de sistemas analógicos com sistemas digitais. A parte analógica é composta pela operação mecânica do trem e seus componentes, tais como circuitos de via e máquinas de chave. Já a parte digital é formada pelo conjunto de sistemas discretos que coordenam e regulam o funcionamento da parte analógica, como a comunicação entre sensores, via software. Em segundo lugar, o sistema modelado é relativamente complexo, mas ainda com características que permitem uma modelagem usando autômatos híbridos, quando dividido em estudos de caso separados.

O funcionamento e a operação da malha metroviária, de um modo geral, estão apoiados em vários componentes e em diferentes equipamentos. Um dos principais aspectos da modelagem é capturar como e quando acontece a cooperação entre as partes que compõem o sistema. Primeiramente, são descritos os equipamentos de via, os quais estão distribuídos ao longo da malha metroviária. Não são apresentados todos os equipamentos que fazem parte do sistema. São descritos somente os equipamentos relevantes para a modelagem e a análise desenvolvidas nos capítulos a seguir.

3.1.1 Conceitos Gerais de uma Malha Metroviária

Alguns dos componentes e conceitos relacionados à operação de uma malha metroviária são sucintamente descritos a seguir:

- CIRCUITOS DE VIA: Circuitos de via são trechos da via onde é possível a detecção da presença de trens, fornecendo indicações de ocupação e de desocupação desses trechos. Essas informações são processadas pelo sistema de sinalização. Outra função do circuito de via, ou simplesmente *cv*, é transmitir códigos de velocidade ao trem, de forma que este trafegue a uma velocidade segura, de acordo com sua posição na via, e de acordo com sua posição relativa a outros trens que estão em movimento na via. Os comandos de restrição de velocidade impõem certos limites à velocidade dos trens, em determinados pontos da via, como por exemplo, entre duas estações ou entre uma estação e o final de via. Os circuitos de via são delimitados por um mecanismo que une dois trilhos eletricamente. Um trem, quando ocupa um circuito de via, impede a recepção de um sinal elétrico, fazendo com que seja detectada sua presença naquele trecho de via.

- MÁQUINAS DE CHAVE: As máquinas de chave dos Aparelhos de Mudança de Via (AMVs) são usadas para desviar a rota de composições. Essas máquinas são controladas por intertravamento e são operadas eletricamente (ou manualmente, em condições de emergência), podendo assumir dois estados: (i) normal; e (ii) reverso. As máquinas de chave também podem, em cada posição, assumir os estados de eletricamente travado ou destravado. Uma máquina de chave está posicionada corretamente quando

existir uma correspondência lógica entre o comando de posicionamento enviado e a informação física da posição da máquina de chave, dada pela ponta da agulha. Ou seja, a informação física deve estar coerente com a informação presente no sistema automático de controle. A máquina de chave está travada quando o respectivo motor está desacoplado e sem alimentação elétrica.

- REGIÕES DE INTERTRAVAMENTO: são as regiões onde se encontram os AMVs. É nessas regiões que se efetuam as mudanças de rota dos trens. São chamadas também de "regiões de AMVs".

- INTERTRAVAMENTO MICROPROCESSADO: faz parte do sistema automático que controla a operação do sistema global, e exerce funções de controle na movimentação dos trens. Este equipamento é encarregado de controlar a movimentação dos trens de forma automática e segura, comunicando-se com as máquinas de chave e com os circuitos de via.

- ZONAS TERMINAIS: são trechos de via, compostos por um ou mais circuitos de via, usados para manobras de retorno dos trens.

As funções básicas do sistema de controle global são: (i) comandar e receber indicações de máquinas de chave; (ii) gerar e verificar códigos de velocidade em circuitos de via; (iii) determinar as indicações de estado (ocupado/desocupado) dos circuitos de via; e (iv) comunicar-se com o intertravamento para verificação de sentido de tráfego, ocupação/desocupação dos circuitos de via adjacentes, e a posição das máquinas de chave. O sistema global comunica-se com as interfaces dos circuitos de via através de Caixas à Margem da Via (CMVs).

O código de velocidade determina a velocidade máxima que um trem pode atingir num certo trecho da via. Existem diferentes códigos de velocidade que podem ser distribuídos de acordo com o segmento de via focalizado. Esses códigos podem variar entre 0 e 100 km/h. A transmissão dos códigos de velocidade num circuito de via é realizada através de um par de CMVs, uma CMV transmissora e outra CMV receptora. Baseado nas informações recebidas do controle de movimentação dos trens, determina-se que velocidade deve ser imposta a um circuito de via em particular. O código é transmitido à CMV daquele circuito que, por sua vez, lhe impõe o código de velocidade adequado.

As chamadas funções de supervisão servem para alimentar o sistema com informações obtidas através da leitura dos estados dos equipamentos da via. Essas informações são utilizadas para implementar o intertravamento, para estabelecer os perfis de velocidade dos trens, e para calcular as curvas de frenagem, entre outras funções.

Os equipamentos que estão na via são comandados através das funções de atuação. A função de intertravamento é um conjunto de regras que permite ao sistema realizar suas

principais atividades, como movimentação e proteção das máquinas de chave e aplicação de códigos de velocidade. O modo de controle do intertravamento indica o modo de operação que deve ser utilizado numa certa região de intertravamento. Existem três modos de controle: o modo central, o modo local e o modo automático. Este último é o modo que interessa para fins de modelagem. No modo automático os comandos sobre os equipamentos de via são gerados automaticamente pelo sistema.

As funções de comando sobre as máquinas de chave permitem uma atuação direta no posicionamento das máquinas de chave nas regiões de AMVs. As máquinas de chave são movimentadas para as posições de normal ou reverso, desde que as condições de intertravamento o permitam. Os AMVs de uma determinada região de intertravamento podem assumir dois modos de controle mutuamente exclusivos, o manual e o automático. O modo de controle manual só é usado em casos de emergência e será desconsiderado na modelagem. Os sinais enviados aos equipamentos de via são responsáveis pelo acionamento e pelo travamento das máquinas de chave.

Entre as funções de controle da movimentação dos trens estão: (i) seleção do modo de controle do intertravamento; (ii) comando sobre máquinas de chave; (iii) comando de restrição de velocidade; e (iv) inversão do sentido de tráfego. A inversão é utilizada quando um ou mais trens ficam bloqueados num trecho de via ou quando um trem chega a uma zona terminal.

3.1.2 Os Requisitos de Segurança de uma Malha Metroviária

Os requisitos básicos relevantes para a modelagem, do ponto de vista de segurança, são evitar colisões e descarrilhamentos de trens. Estas situações podem ocorrer, por exemplo, por uma movimentação errada de um AMV. Por isso, o sistema deve executar de maneira segura as funções de controle de movimentação dos trens, através das funções de proteção dos AMVs, controle de tráfego e controle de seleção dos códigos de velocidade.

São apresentados a seguir os requisitos de segurança que devem ser aplicados ao sistema:

1. Só pode haver a ocupação de um circuito de via em condições normais de operação do sistema se não houver tráfego estabelecido no sentido oposto, e se o circuito de via já não estiver ocupado por nenhum outro trem;

2. Só pode haver a indicação de desocupação de um circuito de via em condições normais de operação do sistema se o circuito de via já estiver desocupado pelo próprio trem;

3. Só pode haver destravamento de uma máquina de chave se o circuito de via a que ela pertence estiver desocupado;

4. O perfil seguro de velocidade em circuitos de via vizinhos a uma ocupação deve ser respeitado pelas ocupações adjacentes;

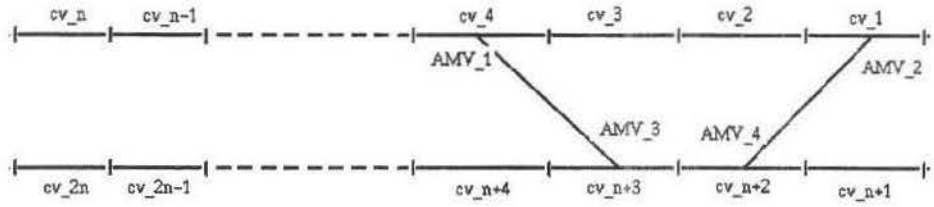


Figura 3.1: Região de vias da malha metroviária

5. Na ocupação seqüencial dos circuitos de via, os códigos de velocidade devem obedecer o sentido do tráfego estabelecido naquela região e os perfis de velocidade impostos pelas condições da via;

6. Só pode haver inversão de sentido se o trecho para onde o trem está se dirigindo não estiver sendo utilizado como saída para outro trem e se o circuito de via a ser invadido não estiver ocupado.

3.1.3 Funcionamento de uma Malha Metroviária

Para oferecer garantias de segurança, sistemas de controle de tráfego metroviário exigem, em função do estado da tecnologia atualmente utilizada, o posicionamento e travamento de um AMV, antes que o trem invada o *cv* onde este AMV está localizado. O modelo tratado, neste trabalho, desvia dessa concepção, conduzindo um exercício de modelagem da dinâmica do sistema onde se permite que o trem adentre ao *cv* antes que o AMV esteja posicionado e travado. Experimentos desse tipo revelam até que ponto o requisito de segurança básico pode ser relaxado, de forma que o sistema ainda ofereça garantias de uma operação segura.

No sistema real podem existir vários trens trafegando pela malha metroviária. Um mapa simplificado de um conjunto de circuitos de via, formando uma região da malha metroviária, pode ser visualizado na Figura 3.1. Os circuitos de via possuem comprimento padrão de cerca de 200 metros. Cada circuito pode ser ocupado para dar passagem a um determinado trem. A ocupação do circuito ocorre só quando este está livre, e respeitando o perfil de velocidade que lhe foi imposto.

A ocupação do circuito é provocada por um pedido de um trem, o qual precisa passar pelo circuito para chegar ao seu destino. A ocupação é concedida impondo-se perfis de velocidade adequados, os quais são enviados aos trens através de transmissores adaptados aos *cvs*. Podem existir diferentes perfis de velocidade atribuídos aos circuitos de via. Os perfis básicos utilizados no sistema são: 0 km/h, 30 km/h, 60 km/h, 80km/h e 100 km/h. A desocupação se dá simplesmente quando o trem deixa o circuito de via no qual estava

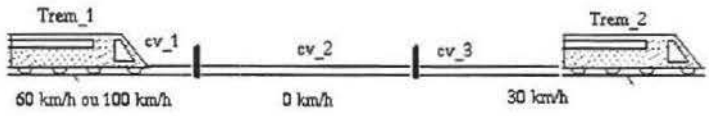


Figura 3.2: Tomada de perfis evitando colisões

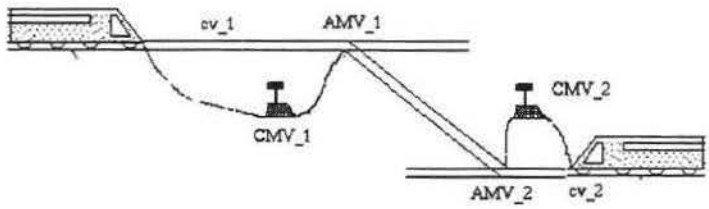


Figura 3.3: Mudança de vias com trens em sentidos opostos

presente, ficando o circuito livre para uma nova ocupação pelo mesmo trem ou para a ocupação por outros trens que realizam aquela mesma rota.

Os perfis de velocidade são impostos de acordo com a posição dos trens na via. Existe uma seqüência de perfis de velocidade pré-determinados que são atribuídos ao longo da via, antecipadamente, tido como perfis padrão. Por exemplo, um circuito que contém um aparelho de mudança de via possui um perfil padrão de 30 km/h. Circuitos de via distantes das regiões de intertravamento podem ter perfis de até 100 km/h. O código de 0 km/h é aplicado a um circuito que, na realidade, não pode ser ocupado. Este circuito pode estar desocupado, mas se o circuito seguinte a ele estiver ocupado, o código de velocidade zero é aplicado ao primeiro circuito. O perfil zero, aplicado a um circuito de via, evita que o trem invada o próximo circuito de via.

Quando se tem a ocupação de um circuito de via, os perfis de velocidade dos circuitos adjacentes são atribuídos conforme sua distância do circuito base. Esse esquema, chamado de *sombra*, garante a segurança da malha, com a imposição de velocidades altas para situações onde não há risco de colisão e velocidades mais baixas onde a probabilidade de colisões seja maior. Veja a ilustração na Figura 3.2. Como os trens estão muito próximos, o perfil atribuído deve ser seguro o bastante para que, no pior caso, o Trem_1 possa parar antes de atingir o trem que está à sua frente. Trafegando a uma velocidade de 60 km/h o Trem_1 consegue parar antes de atingir o Trem_2, que vai a 30 km/h, desde que seja aplicado um código de velocidade de 0 km/h no circuito que vai ser invadido. Se o Trem_1 estiver a 100 km/h ao invadir o cv_2, não conseguirá parar antes de atingir o cv_3, podendo colidir com o Trem_2. Outro tipo de situação que pode ocorrer é ilustrado na Figura 3.3. Quando dois trens trafegam em sentidos inversos, ou seja, quando um trem está percorrendo uma

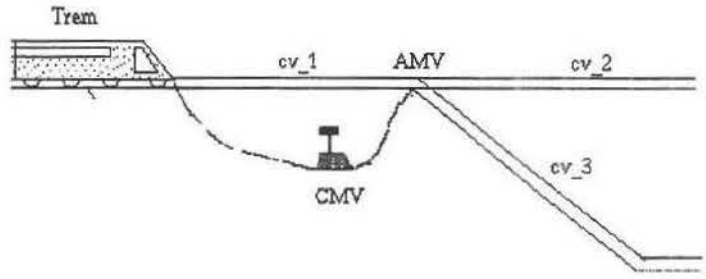


Figura 3.4: Mudança de vias na região de AMV

determinada rota e outro trem está efetuando uma manobra na mesma linha em sentido contrário, existe o perigo, na região do AMV, dos trens trocarem de rota e se chocarem, ocasionando um acidente.

Para que uma composição ocupe uma região de AMV, além do pedido de ocupação do circuito, é necessário que uma requisição de movimentação do AMV seja atendida. Essa movimentação é efetuada conforme a necessidade da ocasião. Observe, na Figura 3.4, que o trem se aproxima e ocupa o circuito de via cv_1. Ao ocupar o circuito cv_1, o trem já deve colocar o pedido de requisição de movimentação do AMV. Se for seguir em frente para o circuito cv_2, o trem deve pedir o posicionamento do AMV para o estado *normal*. Caso queira mudar de direção, indo para o circuito cv_3, o pedido de posicionamento deve colocar o AMV no estado *reverso*. Quando um trem faz o pedido de movimentação para um AMV, este deve se destravar e começar a movimentar-se para a posição correta. Essa movimentação leva algum tempo, até que a ponta da agulha esteja no lugar correto. O tempo de movimentação do AMV deve ser adequado para que, no momento do trem atravessá-lo, o AMV já esteja travado e a travessia ocorra em segurança.

Um outro ponto interessante do funcionamento do sistema é a parte que compreende o fim da via. Essa parte da malha metroviária é ilustrada na Figura 3.1, onde os circuitos de via mais à direita são uma espécie de pátio de manobras para o retorno dos trens. É tomado como padrão o tráfego na via superior no sentido da esquerda para a direita, e na parte inferior no sentido inverso. O trem que ocupa o circuito de via cv_4 pode optar por seguir tanto para o circuito cv_3, como pode optar por trocar de via, indo para o circuito cv_{n+3}. Escolhendo seguir na via superior, o trem deve chegar ao cv_1 e em seguida retornar, no sentido inverso, pedindo o posicionamento do AMV_2 em *reverso*, e partir em direção ao cv_{n+2}. Desse ponto, com o AMV_3 posicionado em *normal*, o trem continua em linha reta, retornando pela linha inferior. Outra possibilidade é o trem que está no circuito cv_4 optar por mudar de via. Nesse caso, o trem teria solicitado a movimentação do AMV_1 e do AMV_3 para *reverso*. Em seguida, o trem ocuparia o circuito cv_{n+3}, prosseguiria até o

circuito cv_{n+1}, para então solicitar o posicionamento do AMV_3 e do AMV_4 em *normal*, e partiria em direção ao circuito de via cv_{2n}.

Atualmente, parte do tráfego metroviário é controlado por sistemas eletrônicos de sinalização, implementados através do uso de microprocessadores. Estes sistemas oferecem facilidades para a diminuição do intervalo entre trens consecutivos na malha, aumentando a capacidade de transporte com níveis compatíveis de disponibilidade, segurança, presteza e confiabilidade, a um custo adequado.

Existem várias outras características do controle do sistema metroviário que podem sofrer mudanças para melhorar o seu funcionamento e sua operação. Uma das opções é a utilização de estações rádio-base [19], onde informações podem ser passadas diretamente para o trem via rádio. Também pode-se considerar a operação automática do trem (*driveless operation*), via rádio, onde o envio e o recebimento de dados é feito diretamente para os equipamentos de via. Com a sinalização via rádio, a distância entre os trens não fica limitada por trechos de via e zonas de proteção. Mais trens podem, então, estar sobre a mesma linha e a freqüência dos trens pode ser maior. Os trens também teriam a responsabilidade de determinar, eles mesmos, suas próprias velocidades seguras, tomando em consideração as condições da via, além da presença e da distância de outros trens.

Outra possibilidade é que os circuitos de via possam ser ocupados dinamicamente, na realização de uma determinada rota. Não seria necessária a ocupação de um conjunto de circuitos de via (alinhamento de rota) para a movimentação numa linha, o que aumenta em muito a disponibilidade, o rendimento e a capacidade do sistema. Define-se apenas um intervalo de circuitos de via para a operação segura do sistema.

As características dos equipamentos, dos componentes, do funcionamento e da operação do sistema metroviário são, portanto, de grande complexidade. São inúmeros circuitos de via e vários AMVs presentes numa malha espalhada por quilômetros. Além disso, é preciso lidar também com a presença de vários trens trafegando simultaneamente por toda a malha metroviária. Esse sistema, com algumas considerações particulares, é estudado nos artigos que compõem os Capítulos 4 e 5.

3.2 Um Sistema de Gerenciamento de Tráfego Aéreo

Os Sistemas de Gerenciamento de Tráfego Aéreo [1] (ATM) são sistemas muito complexos e críticos, com condições operacionais justas e diversos componentes distribuídos, o que torna sua verificação formal difícil de ser realizada. É óbvio que qualquer falha de segurança pode resultar em elevados e intoleráveis danos, tanto em termos de propriedades como de vidas [40]. Diferente do sistema metroviário onde sempre existe a opção de parar os trens em casos

[1] Air Traffic Management System

de emergência, no controle de tráfego aéreo isto não é possível. Portanto, a necessidade de uma verificação formal e rigorosa de todos os requisitos de segurança do sistema é evidente, principalmente na verificação de softwares de controle de tráfego aéreo.

3.2.1 Controle de Tráfego Aéreo

Em um vôo padrão, após o avião entrar em uma rota, o Controle de Tráfego Aéreo[2] (ATC) monitora o vôo, via rádio [22]. Uma vez contatado, um ATC reconhece o contato do avião e lhe transmite dados de vôo, tais como vetores de direção e valores de altura. Dependendo do plano de vôo, um avião pode alternar muitas vezes, de um controlador de rota para outro, durante seu vôo. Os ATCs são atribuídos à áreas geográficas específicas, e trabalham para manter distância segura entre aviões que cruzam seu setor de espaço aéreo.

Enquanto todos os aviões de linhas aéreas regulares são controlados passo a passo, o mesmo nível de controle positivo não se aplica sempre a todos os aviões, em geral. Alguns aviões podem, e freqüentemente o fazem, voar em espaço aéreo não controlado, fora do alcance do ATC. Em geral, estes espaços não controlados são áreas abaixo das linhas de cruzeiro usadas pelos aviões de linhas aéreas regulares.

Com tempo bom e visibilidade adequada, é permitido se voar sob regras visuais de vôo, ou VFR[3]. Nestes casos, não é necessário registrar um plano de vôo, e não é preciso estar em contato com o controle de tráfego aéreo, a menos que os pilotos entrem no espaço aéreo de um aeroporto que opere uma torre de controle. Sob regras de VFR, os pilotos são responsáveis por manter a separação adequada de outros aviões. Por isso, estas regras, às vezes, são chamadas regras de "ver e ser visto".

Regras para operar com instrumentos de vôo, ou IFR[4], por outro lado, são regras às quais os pilotos, em geral, devem se submeter quando o tempo está ruim ou a visibilidade é baixa. Nestes casos, os pilotos devem estar em contato com o ATC e devem registrar um plano de vôo. Eles também devem ser "pilotos certificados", ou seja, devem estar habilitados para navegação e vôo por instrumentos e sob visibilidade nula. Vôos de linhas aéreas regulares sempre operam sob instrumentos, sem levar em consideração o tempo, e desde que operem dentro da abrangência do sistema ATC.

Avanços tecnológicos permitiram introduzir funções sofisticadas em sistemas de controle de tráfego aéreo, tais como funções de navegação e funções de monitoramento da separação entre aviões [41]. Esta tecnologia mais avançada permite reduzir o número de situações de colisão, ainda mantendo o *throughput* do espaço aéreo, e com níveis adequados de segurança e de confiabilidade. O uso de tecnologias mais avançadas, por outro lado, abre possibilidades para se introduzir erros sutis nos sistemas de controle e monitoramento. Nestes casos, uma

[2] Air Traffic Control
[3] Visual flight rule
[4] Instrument flight rule

verificação formal da segurança do sistema torna-se mais do que desejável, passa a ser mandatória.

3.2.2 Sistema de Prevenção de Colisão e Alerta de Tráfego

Com o passar dos anos, o tráfego aéreo continuou a crescer cada vez mais. Os desenvolvimentos de modernos sistemas de controle de tráfego aéreo tornaram possíveis lidar com este crescimento, ainda mantendo os níveis necessários de segurança de vôo. Entretanto, o risco de colisão no transporte aéreo ainda permanece. O desenvolvimento inicial do Sistema de Prevenção de Colisão e alerta de Tráfego[5] (TCAS) veio para reduzir ainda mais o risco de colisões no tráfego aéreo, mesmo sob a pressão adicional do aumento no número de aviões em vôo [40].

O TCAS é um equipamento usado para detectar e evitar conflitos entre aviões. Cada avião leva a bordo um TCAS. Sua tarefa é monitorar o tráfego aéreo na vizinhança do avião. Uma outra de suas funções é prover o piloto com informações a respeito dos aviões nas proximidades e que possam colocá-lo numa posição de risco de colisão. Outra das funções do TCAS é sugerir como resolver conflitos de rotas entre aviões.

O TCAS melhora significativamente a segurança do vôo, sendo projetado para garantir ausência de colisão entre dois aviões com velocidade de aproximação de até 1200 nós e variações verticais tão altas quanto 10000 pés por minuto[6]. Contudo, não é eliminado inteiramente todos os riscos de colisão pois, como em qualquer sistema preditivo, um risco de colisão pode ser induzido na operação do sistema [34].

Na operação do TCAS, o sistema pode entrar em um, de dois níveis de vigilância existentes [37]. No primeiro nível, o sistema emite um aviso de tráfego[7] (TA), sinalizando ao piloto uma potencial ameaça. Mas, neste caso, não fornece quaisquer sugestões sobre como resolver a situação. Se o risco de colisão aumentar, um aviso de resolução[8] (RA) é emitido, e o sistema sugere também uma manobra para resolver o conflito.

Neste trabalho, o TA é ignorado e, quando a situação de conflito aparece, um RA é emitido diretamente. Esta decisão foi tomada para manter a complexidade dos modelos dentro de limites aceitáveis, de modo que a ferramenta HyTech pudesse completar as análises dos modelos construídos. Além disso, neste trabalho, os RAs emitidos pelo TCAS estão restritos somente ao plano vertical, isto é, as manobras envolvem apenas a subida ou a descida de aviões. Isso significa que as aproximações paralelas não são estudadas, mas somente as verticais o são. Esta decisão também foi tomada tendo em vista a complexidade dos modelos gerados, quando contrastada com os recursos computacionais disponíveis para a

[5] Traffic alert and Collision Avoidance System
[6] 1 pé $\approx$ 0.3048 metros; 1 pé por minuto $\approx$ 0.0051 metros por segundo
[7] Traffic Advisory
[8] Resolution Advisory

realização das simulações. Além disso, hoje, ainda poucos trabalhos estudam o problema da separação horizontal [42, 15].

O sistema TCAS também permite comandos de reversão. Isto é, o TCAS pode mudar o aviso de subida/descida, no decorrer das manobras. Esta última característica foi adicionada ao sistema TCAS para compensar o não-determinismo na atitude do piloto. Se o piloto escolhe não seguir o aviso fornecido pelo TCAS, este detecta o fato e muda o RA, se necessário. Obviamente, este não-determinismo introduz ainda mais complexidade no sistema, e reforça a necessidade de uma verificação e validação formal da segurança do sistema como um todo.

3.2.3 Um Estudo de Caso com Dois Aviões

A ênfase deste estudo está na análise e verificação de aspectos de segurança de um sistema de gerenciamento de tráfego aéreo. O objetivo é modelar o sistema, verificar sua operação e sintetizar valores mais justos para alguns parâmetros críticos do sistema, de tal maneira a melhorar seu desempenho, ainda reduzindo os riscos de operação insegura dos aviões em vôo. Um dos principais aspectos da modelagem é capturar como e quando ocorre a cooperação entre as partes do sistema, ajustando o controle para permitir operações mais seguras.

A Figura 3.5 descreve a situação de dois aviões viajando num mesmo espaço aéreo. As mudanças de atitude para os aviões são provocadas pelo controlador que monitora as

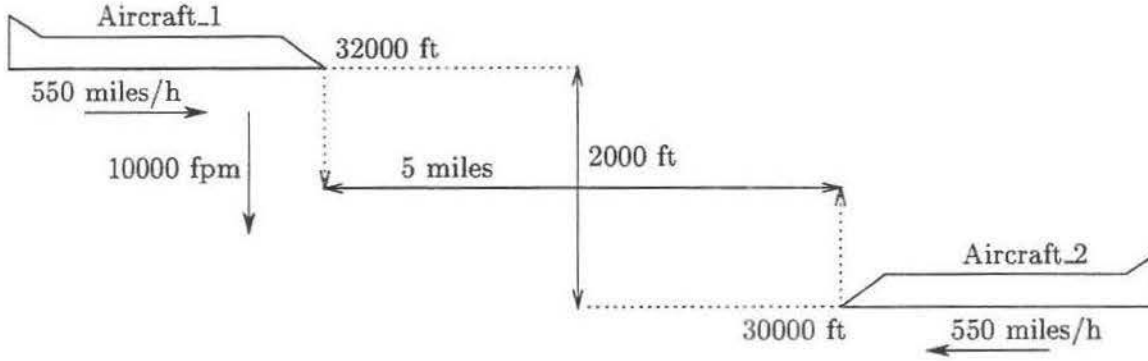


Figura 3.5: Dois aviões no mesmo espaço aéreo

separações verticais e horizontais. Quando um avião quer mudar sua rota ou quando quer simplesmente aterrizar, ele deve obedecer algumas regras de segurança. A obediência à essas regras é garantida pelos comandos emitidos pelo controlador. O controlador pode emitir mensagens para o avião descer ou subir. De acordo com a posição entre os aviões, o controlador pode emitir mensagens para um avião aumentar a descida ou reduzir sua

velocidade. No modelo considerado aqui, com dois aviões, o primeiro avião pode aumentar sua velocidade de descida ou arremeter, dependendo da sua distância em relação ao segundo avião. O segundo avião, por sua vez, pode reduzir sua velocidade para que o primeiro avião possa cruzar sua rota a uma distância segura.

Os padrões da separação dos aviões variam de acordo com as circunstâncias. Acima de 29000 pés, quando os aviões estão viajando em alta velocidade, o padrão é cinco milhas de separação horizontal e 2000 pés de separação vertical. Abaixo de 29000 pés a separação vertical é reduzida para 1000 pés, enquanto que a separação horizontal permanece em cinco milhas. Quando os aviões estão se movendo em velocidades mais lentas, como por exemplo quando partem ou se aproximam de um aeroporto, o padrão é três milhas de separação horizontal e 1000 pés de separação vertical. A ação do controlador começa quando a separação horizontal dos aviões é de 5 milhas, ou menos. O valor padrão para a subida e para a descida vertical é 10000 pés por minuto. Nas emergências, esta velocidade pode ser elevada para 12000 pés por minuto.

No caso ilustrado na Figura 3.5, um avião está viajando numa altura de 32000 pés, da esquerda para a direita e o outro avião está viajando no sentido oposto, a uma altura de 30000 pés. Supõem-se os aviões viajando a 550 milhas por hora, em cruzeiro. Durante a operação do sistema, a velocidade de 490 milhas por hora pode ser aplicada a fim de reduzir a velocidade de cruzeiro numa situação crítica. Estas taxas e valores são baseados em dados reais para Boeings 707 e Boeings 747, com possíveis variantes conforme a situação de vôo[9].

O modelo do autômato do avião que está voando da esquerda para a direita é mais complexo que o modelo do autômato do outro avião, o qual está voando na direção oposta. Isto se dá porque, em alguns instantes, o primeiro avião pode decidir envolver-se em algum tipo de manobra específica, enquanto assume-se que o segundo avião permanece em rota de cruzeiro. O modelo integral do sistema compreende três autômatos híbridos individuais, um para cada avião e um outro para o controlador TCAS/ATC. Todos os estudos de caso relatados no Capítulo 6 focalizam a segurança do sistema. Para cada caso, é conduzida uma análise do modelo, e é também realizada uma síntese paramétrica para sintetizar valores críticos, tal como a altura relativa e a separação horizontal dos aviões.

Vale notar que, para abordar um cenário mais realista, com vários aviões e com manobras mais complexas, os autômatos híbridos apresentariam mais modos de operação distintos. O autômato produto, por conseguinte, resultaria bem mais complexo, excedendo a capacidade computacional da ferramenta HyTech, que já operava no limite de suas possibilidades, num PC típico, de 350MHz.

Uma extensão deste trabalho, portanto, poderia ser direcionada para portar o código da ferramenta para máquinas de maior capacidade. Uma vez portado seu código, a ferramenta HyTech poderia ser utilizada para analisar cenários mais complexos e mais realistas.

[9] http://www.air-transport.org/public/Handbook/

Capítulo 4

Análise e Verificação de Segmentos de Via de uma Malha Metroviária

Prólogo

Uma malha metroviária é composta por quilômetros de via, nas quais trafegam inúmeros trens, além de possuir vários outros componentes mecânicos que fazem parte do sistema, como as máquinas de chave. A operação e o funcionamento de todas essas peças juntas é de grande complexidade. Mesmo assim, todos os componentes distribuídos do sistema devem trabalhar de maneira coordenada, de modo que o comportamento global do sistema resulte seguro.

Devido a essa complexidade, é necessário aplicar ao sistema um processo formal de avaliação, que garanta uma operação segura. Para modelar sistemas com estas características, a abordagem usando autômatos híbridos como técnica formal tem se mostrado eficaz. Este capítulo é composto pelo artigo, "Análise e Verificação de Segmentos de Via de uma Malha Metroviária", que explora a técnica de autômatos híbridos num estudo realizado sobre um segmento de via de uma malha metroviária. Várias propriedades de segurança da operação do segmento de via são verificadas, com o objetivo de validar o funcionamento do sistema. O artigo foi publicado nos anais do *2nd Workshop on Formal Methods* (WFM'99), realizado em Florianópolis, Santa Catarina, de 12 a 13 de outubro de 1999.

O estudo desenvolvido neste trabalho está voltado para um sistema metroviário genérico. Informações mais detalhadas a respeito de sistemas metroviários específicos podem ser encontrados em [14, 19, 10].

Análise e Verificação de Segmentos de Via de uma Malha Metroviária

Adilson Luiz Bonifácio*, Arnaldo Vieira Moura†, João Batista Camargo Jr.‡, Jorge Rady de Almeida Junior§

30 de agosto de 1999

Resumo

O objetivo desse trabalho é aplicar técnicas de especificação formal para modelar sistemas distribuídos realistas, certificando que atendem a várias normas de segurança. Os modelos formais são baseados em autômatos híbridos. Os alvos desse trabalho são segmentos de via de uma malha metroviária. O modelo construído é analisado e certificado, garantindo que o sistema estudado opera de forma segura. A análise e a certificação do modelo são auxiliadas por ferramentas computadorizadas semi-automáticas.

Palavras-Chave: Análise, Síntese, Sistemas Híbridos, Sistema Metroviário, Verificação.

1 Introdução

Linguagens formais e modelos matemáticos promovem a eliminação de inconsistências e ambigüidades muitas vezes encontradas em especificações de projetos [5]. O uso de formalismos para especificação de sistemas tem se tornado cada vez mais indispensável no desenvolvimento de sistemas distribuídos reativos, que operam em tempo real, onde uma falha de operação causaria danos irreparáveis. Neste trabalho, especificações formais são

*Instituto de Computação, Universidade Estadual de Campinas, Caixa Postal 6176, Campinas, SP, 13081-970, albonifa@dcc.unicamp.br - Suporte parcial da CAPES, DS - 44/97

†Instituto de Computação, Universidade Estadual de Campinas, Caixa Postal 6176, Campinas, SP, 13081-970, arnaldo@dcc.unicamp.br - FAPESP - 98/05999-4

‡Departamento de Engenharia de Computação e Sistemas Digitais, Escola Politécnica, Universidade de São Paulo, Av. Prof. Luciano Gualberto, Trav. 3, N. 158, São Paulo, SP, 05508-900, jbcjunio@pcs.usp.br

§Departamento de Engenharia de Computação e Sistemas Digitais, Escola Politécnica, Universidade de São Paulo, Av. Prof. Luciano Gualberto, Trav. 3, N. 158, São Paulo, SP, 05508-900, USP, jrady@pcs.usp.br

usadas para modelar um sistema dessa natureza, formado por segmentos de uma malha metroviária.

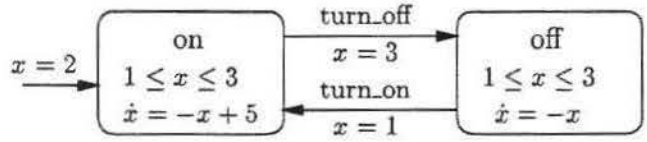
Usualmente, as teorias clássicas de controle de sistemas discretos lidam com poucos componentes distribuídos cujas características dinâmicas são complexas. Este estudo, lida com sistemas que apresentam vários componentes distribuídos com características discretas determinantes e características dinâmicas mais simples. Autômatos híbridos [1] são uma das formas de se especificar tais sistemas. Esse formalismo estende a noção de autômatos finitos, agregando aos estados equações que determinam a dinâmica do sistema e associando às transições entre estados mudanças no seu perfil dinâmico. Os diferentes componentes do sistema são modelados por autômatos independentes, sendo o comportamento global do sistema obtido da cooperação desses autômatos. Em geral, na prática, sistemas induzem um modelo de uma complexidade tal que se torna imprescindível o suporte de ferramentas computacionais para análise da especificação produzida. Neste trabalho foi utilizada a ferramenta HyTech [8], que lida com autômatos híbridos lineares.

As próximas seções estão organizadas da seguinte forma. A seção 2 define autômatos híbridos e apresenta a ferramenta computacional. A seção 3 descreve a malha metroviária analisada. A seção 4 apresenta a análise e verificação dos modelos construídos. A última seção apresenta as conclusões finais a respeito do trabalho.

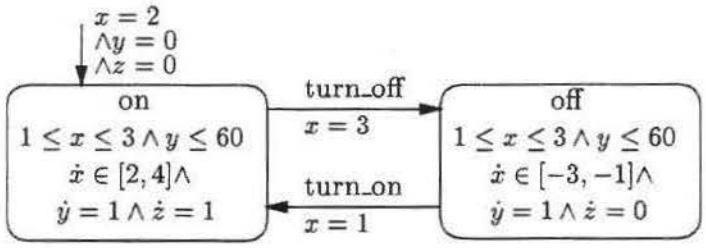
2 Autômatos Híbridos

A Figura 1(a) descreve a temperatura x no interior de um aquecedor [11]. O modelo é constituído por dois modos de operação: *on* e *off*. A condição inicial em *on* é $x = 2$ graus e, nesse modo de operação, o perfil dinâmico do sistema é descrito por $dx/dt = -x + 5$, ou seja, quando o aquecedor está ligado a temperatura aumenta de acordo com a função diferencial $dx/dt = -x + 5$.[1] O sistema só se mantém em um modo de operação enquanto satisfizer a condição invariante associada ao modo de operação. No exemplo, a condição invariante para ambos os modos de operação é $1 \leq x \leq 3$, isto é, a temperatura se mantém entre 1 e 3 graus. Uma mudança no perfil dinâmico do sistema é caracterizada por uma transição entre modos de operação, habilitada por uma condição que especifica também novos valores para as variáveis reais e pode, ainda, emitir uma mensagem de sincronismo. Na transição de *on* para *off* a condição de habilitação é $x = 3$, o valor de x não é alterado, o perfil dinâmico muda para $dx/dt = -x$, e a mensagem emitida é *turn_off*. A condição $x = 3$ indica que, ao chegar em 3 graus, o aquecedor deve ser desligado e sua temperatura passa a decrescer obedecendo o novo perfil $dx/dt = -x$. Na transição, o valor de x não se altera. O comportamento do sistema é descrito pela evolução das variáveis reais ao longo

[1] Usaremos também a notação $\dot{x}$ em substituição à dx/dt.



(a) Aquecedor simples



(b) Aquecedor e requisito de segurança

Figura 1: Autômatos híbridos: um exemplo

do tempo, a partir da condição inicial.

Um autômato híbrido é um sistema $A = (X, V, init, inv, flow, E, jump, \Sigma, syn)$ onde X é um conjunto de variáveis e V é um conjunto de modos de operação. Nesse sistema, $init(v)$ dita a condição inicial, $inv(v)$ representa a condição invariante, e $flow(v)$ descreve o perfil dinâmico do modo de operação v, para todo $v \in V$. O item E é um multi-conjunto de transições definidas em $V \times V$. O item $jump(e)$ define a condição e os novos valores das variáveis reais para a transição e. O item Σ é um conjunto de eventos e $syn(e)$ associa um evento à cada transição e, para todo $e \in E$. O formalismo usa o símbolo x' para denotar o novo valor da variável x após a mudança de modo de operação. Na Figura 1(a), se $e = (on, off)$ então $jump(e)$ seria $x = 3 \wedge x' = x$ e $syn(e)$ seria $turn\_off$. Como $x' = x$ indica que o valor de x não é alterado, no modelo gráfico é indicada apenas a condição de mudança $x = 3$. Uma descrição formal mais detalhada pode ser encontrada em [1, 6, 11].

Em geral, cada componente de um sistema é modelado por um autômato híbrido independente e o comportamento global do sistema é capturado pelo autômato produto [6, 8, 11]. Os autômatos independentes cooperam: (i) usando variáveis compartilhadas, e (ii) sincronizando eventos. Sejam A_1 e A_2 dois autômatos com transições e_1 e e_2. Essas transições são simultâneas no autômato produto se $syn_1(e_1) = syn_2(e_2)$. São intercaladas se $syn_1(e_1) \neq syn_2(e_2)$, $syn_1(e_1) \notin \Sigma_2$, e $syn_2(e_2) \notin \Sigma_1$.

Uma *configuração* é um par (v, a), onde v é um modo de operação e a é um vetor de

valores para as variáveis reais. A configuração (v, a) é *admissível* se o predicado *inv(v)* é verdadeiro, e é *inicial* se a condição *init(v)* é satisfeita, quando as variáveis reais são substituídas pelos seus correspondentes valores em a. Uma *trajetória* é uma seqüência de configurações que obedecem à dinâmica do modelo. Uma configuração é *alcançável* se ocorre em alguma trajetória que parte de uma configuração inicial. Na Figura 1(a), $(on, 1.5)$ é uma configuração admissível, $(on, 2)$ é uma configuração inicial, e toda configuração admissível é alcançável. O conjunto de configurações para as quais um predicado φ é satisfeito é denominado φ-região, ou região φ. A região *inv* é formada por todas as configurações admissíveis. Da mesma forma, a região *init* agrupa todas as configurações iniciais. Dada uma região φ, $Post(\varphi)$ reúne todas aquelas configurações q' para as quais existe uma configuração q, contida na região φ, e tal que a configuração q' é alcançável a partir de q através de uma trajetória de A constituída de uma atividade contínua, seguida ou não de uma mudança de fase. Partindo de uma região φ inicial, a sucessiva aplicação do operador *Post*, quando converge, produz a região alcançável, a partir da região φ, através de trajetórias do autômato A. No outro sentido, dada uma região φ, $Pre(\varphi)$ contém uma configuração q se existe uma configuração q', na região φ, tal que q' é alcançável a partir de q através de uma trajetória de A formada por uma atividade contínua, seguida ou não de uma mudança de fase.

Um autômato híbrido A é *linear* se *inv(v)*, *init(v)* e *jump(e)* são predicados lineares convexos, e se *flow(v)* é um predicado linear convexo sobre as variáveis de $\dot{X}$. Um *predicado linear convexo* é uma conjunção finita de inequações envolvendo combinações lineares racionais de variáveis, tal como $2x + 7z/2 \leq -10$. Pode-se mostrar o seguinte teorema: se A é um autômato híbrido linear, e se φ é uma disjunção finita de predicados lineares convexos, então a iteração do operador *Post*, a partir da região definida pelo predicado φ, converge para uma nova região de configurações de A [1]. Esse teorema permite automatizar a análise de modelos baseados em autômatos híbridos lineares.

Um modelo envolvendo autômatos híbridos é dito *seguro* se as configurações alcançáveis satisfazem certos requisitos de segurança. Geralmente, um requisito de segurança é especificado na forma de um predicado que separa as configurações do autômato em configurações desejáveis e configurações indesejáveis. Um requisito de segurança para o aquecedor poderia ser: não permanecer ligado mais de 2/3 do tempo nos primeiros 60 minutos de operação. Na Figura 1(b), a variável y mede o tempo total e a variável z mede o tempo que o sistema permanece no modo *on*, ou seja, ligado. Portanto, uma configuração é *insegura* se satisfaz o predicado linear convexo $y = 60 \wedge z > 2y/3$. Neste modelo, o perfil dinâmico no modo de operação *on* foi relaxado para $\dot{x} \in [2, 4]$, pois sabe-se que nesse modo de operação vale $\dot{x} = -x + 5$ e $1 \leq x \leq 3$. O modo *off* é relaxado de forma similar. A relaxação resulta num autômato linear que, se demonstrado seguro, garante também a segurança do sistema original.

Em geral, o exercício de modelagem produz autômatos de uma complexidade tal que se faz necessário o suporte de ferramentas computacionais automáticas. O software HyTech [9, 11, 10, 8] é uma ferramenta para auxiliar na análise e verificação de modelos híbridos lineares. O arquivo de entrada possui duas partes. A primeira parte descreve os autômatos híbridos participantes. O autômato produto é calculado automaticamente. A segunda parte possui uma seqüência de comandos de análise. A linguagem de especificação desses comandos lembra uma linguagem de programação, com tipos de dados e primitivas. Vários exemplos de uso da ferramenta podem ser encontrados em [1, 11, 10, 8]. A ferramenta HyTech é capaz de verificar, automaticamente, que nenhuma configuração alcançável, a partir da configuração inicial, satisfaz à condição de insegurança para o modelo do aquecedor ilustrado na Figura 1(b). Esse resultado estabelece que o sistema é seguro.

O HyTech, a princípio, foi usado para analisar sistemas de controle, como é o caso do exemplo do aquecedor. Hoje, a ferramenta também está sendo usada, inclusive, na modelagem e análise de circuitos temporizados. O HyTech é, ainda, a única ferramenta automática de apoio a análise baseada em autômatos híbridos. Existem outras ferramentas, como o UPPAAL [15, 3, 4, 2], baseadas em *timed-automata*, uma classe mais restrita dos autômatos híbridos lineares, e que trabalham apenas com relógios, não suportando equações diferenciais mais complexas em seus perfis dinâmicos. Inúmeros outros trabalhos relacionados podem ser encontrados em [14, 12, 13].

3 Descrição da Malha Metroviária

Atualmente, o tráfego metroviário é controlado por sistemas eletrônicos de sinalização implementados através do uso de microprocessadores. Estes sistemas oferecem facilidades para a diminuição do intervalo entre trens consecutivos na malha, aumentando a capacidade de transporte com níveis compatíveis de segurança [7]. Basicamente, circuitos analógicos implementam sensores e atuadores, e circuitos digitais microprocessados controlam o intertravamento de segurança de todo o sistema. Um sistema desse porte é complexo no seu todo, mas segmentos da malha são passíveis de modelagem usando autômatos híbridos. Um dos principais aspectos da modelagem é capturar a cooperação entre as partes componentes do sistema, além de garantir sua segurança.

Circuitos de via, ou *cvs*, são trechos da via onde é possível detectar a presença de trens. O comprimento padrão dos *cvs* é de cerca de 200 metros. A ocupação do *cv* é provocada pela presença do trem e a desocupação se dá quando o trem abandona o *cv*. Perfis de velocidade são enviados aos trens através de transmissores adaptados aos *cvs*, permitindo com que os trens trafeguem de forma segura. Os perfis de velocidade básicos são 0, 30, 60, 80 e 100 km/h. Na ocupação de um *cv*, os perfis atribuídos a *cvs* adjacentes são graduados conforme sua distância do *cv* base. Esse esquema, chamado de *sombra*, garante a segurança

da malha. O perfil de velocidade zero também indica que um *cv* não pode ser invadido por um trem.

Os Aparelhos de Mudança de Via (AMVs) permitem com que os trens troquem de linha. O pedido de posicionamento de um AMV provoca sua movimentação e posterior travamento. A movimentação consome algum tempo, até que a ponta da agulha esteja em posição e o AMV trave. A velocidade do trem e o tempo de movimentação do AMV devem ser adequados para que o trem passe pelo AMV em segurança. Os AMVs podem assumir duas posições: normal e reverso. Em cada posição, eles podem assumir os estados de travado ou destravado. Os equipamentos de intertravamento microprocessado exercem funções de controle dos trens, como, por exemplo, o comando de restrição de velocidade e a inversão no sentido de tráfego. Os comandos de restrição de velocidade impõem limites à velocidade dos trens. O comando de inversão permite inverter o sentido do tráfego entre regiões de AMVs.

O requisito de segurança básico é evitar colisões e descarrilhamentos. Algumas condições de segurança são: (i) só ocupar um *cv* se não houver tráfego no sentido oposto; (ii) só ocupar um *cv* se este estiver desocupado; (iii) só indicar a desocupação de um *cv* quando o trem o abandonar; (iv) só destravar um AMV em um *cv* desocupado; (v) respeitar o perfil de velocidade em *cvs* anteriores e posteriores a um *cv* ocupado; (vi) na ocupação seqüencial dos *cvs*, os códigos de velocidade devem obedecer ao sentido do tráfego e aos perfis de velocidade; e (vii) só inverter o sentido se o *cv* alvo estiver desocupado. Mais detalhes podem ser encontrados em [6].

Nas zonas terminais da malha os trens manobram para inverter o sentido do percurso. O segmento de malha modelado representa um pátio de manobras, e inclui cinco *cvs*, além de dois AMVs, como ilustrado na Figura 2.

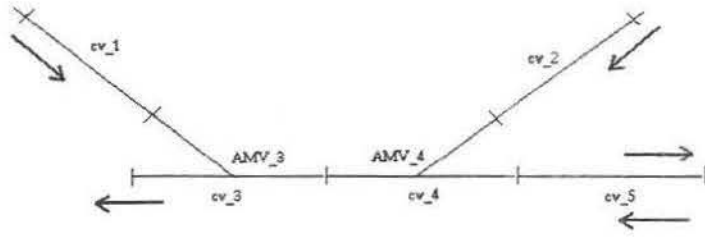


Figura 2: Segmentos de vias utilizados em um pátio de manobras

4 Modelagem e Verificação da Malha

Para oferecer garantias de segurança, muitos sistemas de controle de tráfego metroviário exigem o posicionamento e travamento de um AMV, antes que o trem invada o *cv* onde este AMV está localizado. *O modelo tratado neste trabalho desvia dessa concepção, conduzindo um exercício de modelagem da dinâmica do sistema onde se permite que o trem adentre ao* cv *antes que o* AMV *esteja posicionado e travado. Experimentos desse tipo revelam até que ponto o requisito de segurança básico pode ser relaxado e de forma que o sistema ainda ofereça garantias de uma operação segura.*

O modelo completo do sistema, ilustrado na Figura 2, é formado por: dois autômatos híbridos, um para cada trem que trafega pela malha; mais dois autômatos, um para cada AMV; além de outros cinco autômatos, um para cada um dos *cvs*. O autômato do trem possui oito estados, usados para representar a circulação do trem pelos *cvs* e a passagem pelos AMVs. Veja a ilustração na Figura 3(a). As variáveis usadas são $aux1$ e k, para um dos trens, e $aux2$ e w, para o outro trem. As variáveis $aux1$ e $aux2$ indicam a distância dos trens dentro dos *cvs* ocupados. As variáveis k e w incorporam as regras de ocupação dos *cvs*. O autômato inicialmente se encontra no estado "Ocupar", na iminência de ocupar um dos *cvs*, no caso o cv_1, se este estiver desocupado e não existir ocupação cedida a outro trem no cv_3. Após a ocupação efetiva do *cv*, o trem começa a se locomover a uma velocidade que varia de 7 a 9 metros por segundo, mantém essa velocidade até atravessar os 200 metros do cv_1, para em seguida ocupar o cv_3. Se este estiver ocupado, o trem espera até que seja liberado. O trem só ocupa o cv_3 se o cv_4 e o cv_5 estiverem livres. Quando ocupa o cv_3, o trem pede o posicionamento do AMV_3 para *reverso*, atravessa o AMV_3 e continua no cv_3. Antes de deixar o cv_3, o trem requisita a ocupação do cv_4. Ao entrar no cv_4, o trem pede a movimentação do AMV_4 para *normal* e completa a travessia do cv_4. No cv_5 a operação se repete e o trem percorre toda a sua extensão. No final do cv_5 o trem inverte o sentido do percurso, recebendo um perfil de velocidade no intervalo de -9 a -7 metros por segundo. No outro extremo do cv_5 o trem só ocupa o cv_4, se este estiver livre. Ao entrar no cv_4, o trem pede o posicionamento do AMV_4 para *normal*, atravessa a região do AMV_4 e libera o cv_4, passando a ocupar o cv_3 quando pede a movimentação do AMV_3 para *normal*. Após percorrer o cv_3, o trem desocupa este circuito e retorna ao ponto inicial. Se decidir manobrar pelo cv_2, o trem só pode ocupar o cv_4 caso não haja outro trem no cv_3, no sentido inverso, ou no cv_4. Quando ocupar o cv_4, o trem requisita a movimentação do AMV_4 para *reverso*, passa pelo AMV_4, prossegue para o fim do cv_4 e o libera. Finalmente, o trem ocupa o cv_3, pede o posicionamento do AMV_3 para *normal*, atravessa o cv_3 e o libera. Novamente, o trem retorna para o ponto inicial e recomeça a simulação. Note na Figura 3(a) que os estados "Ocupando" do autômato identificam a ocupação dos diferentes *cvs* no segmento de via, numerados de 1 até 5, de acordo com a variável de controle k. Os

dois estados denominados "Perfil" denotam as velocidades exercidas na região de AMVs. Um desses perfis indica velocidade positiva, no sentido normal da via (da esquerda para a direita), e o outro perfil indica velocidade negativa, no sentido contrário. A tomada dos perfis é regida pelas variáveis k e w. Quando um *cv* é desocupado o controle do autômato passa para o estado "Ocupar" novamente, permanecendo na iminência de ocupar outro *cv*.

O autômato do AMV é constituído por quatro estados como mostra a Figura 3(b). As variáveis usadas são $t3$ e $t4$, uma para cada um dos dois AMVs. As variáveis $t3$ e $t4$ indicam o tempo de movimentação dos AMVs. O autômato é iniciado na posição *normal* e o AMV começa a se deslocar assim que há um pedido de movimentação. O tempo normal de movimentação é de 15 segundos, sendo tolerada uma imprecisão para menos de até 20% nesse valor. Dentro de 12 a 15 segundos o AMV estará travado na posição *reversa*. A movimentação do AMV no sentido contrário, a partir do estado *reverso*, é similar. Da figura, pode-se notar que as variáveis são iniciadas em 0 (zero) e que o sistema finaliza a movimentação e posterior travamento em 15 segundos. Porém, o tempo varia de acordo com o avanço de um relógio que admite uma deriva de até 20%. Nos autômatos dos AMVs, os eventos "Abrir" e "Fechar" sincronizam com os mesmos eventos nos autômatos dos trens, no momento dos pedidos de posicionamento.

Cada *cv* é modelado por um autômato que indica sua ocupação e liberação, como ilustra a Figura 3(c), e cada autômato de um *cv* possui apenas os estados "Desocupa" e "Ocupa". O autômato híbrido que modela o sistema completo é obtido pelo produto de todos os nove autômatos independentes. A ferramenta HyTech calcula esse autômato produto automaticamente.

O desenvolvimento do modelo completo apresentado nesse trabalho demandou um custo relativamente alto em termos do tempo total despendido na modelagem, considerando-se que o modelo é relativamente simples. Parte deste custo deve ser atribuído à limitações da ferramenta e à escassez de recursos computacionais disponíveis. Em primeiro lugar, sendo o HyTech ainda uma ferramenta acadêmica, os manuais de utilização não são adequados. Em segundo lugar, a ferramenta possui algumas lacunas entre a sua implementação e a sintaxe de sua linguagem, apresentada na documentação disponível. Essas lacunas demandaram algum tempo até serem removidas, através de experimentos práticos desenvolvidos especialmente para este fim. Uma outra dificuldade encontrada no desenvolvimento do trabalho diz respeito ao consumo de recursos computacionais, por parte da ferramenta. Algumas idéias de modelagem foram descartadas devido a este limitador. Para contornar essa dificuldade adotou-se a estratégia de uma construção *bottom-up* e cuidadosa dos modelos, buscando sempre trabalhar no limite operacional da ferramenta e dos recursos computacionais disponíveis. Uma outra alternativa para evitar os limites no consumo de recursos computacionais foi segmentar o sistema em componentes menores, passíveis de modelagem. A partir desses modelos várias propriedades foram analisadas de modo a garantir

a segurança da operação do segmento de malha selecionado. Nas análises conduzidas, a ferramenta executou sempre numa plataforma Pentium II, de 350 MHz, 320 MB de RAM, e 420 MB para *swap*. Foram analisadas propriedades que dizem respeito à posição relativa dos trens na via, e também foram consideradas propriedades relativas à movimentação dos AMVs quando da passagem dos trens pelos circuitos de via que contém esses AMVs.

Quanto à posição relativa dos trens as situações que levam o sistema a uma operação insegura são: (i) um trem ocupando o cv_3, e outro trem ocupando o cv_4 ou o cv_5, em qualquer direção; (ii) um trem ocupando o cv_2 com outro trem no cv_4, na mesma rota; e (iii) um trem ocupando o cv_1, e outro trem ocupando o cv_3, na mesma direção. Os resultados da verificação da segurança do sistema quanto a esse item são apresentados na Tabela 1. Nessa verificação, o pedido de movimentação dos AMVs se dá na posição zero do *cv*. A primeira e a segunda colunas indicam a posição e o sentido do primeiro trem. A terceira e a quarta colunas repetem esta informação para o segundo trem. A quinta coluna classifica a situação como *safe* ou *unsafe*. Para efeitos de segurança, uma região de configurações *safe* pode ser atingida por alguma trajetória que parte da região de configurações iniciais do modelo. Uma região do tipo *unsafe*, se alcançada, representa uma violação de segurança. A ferramenta HyTech calcula as configurações alcançáveis do modelo e determina se alguma delas é *unsafe*. A última coluna da tabela apresenta a indicação da ferramenta. Uma inspeção da tabela revela que não foram detectadas violações de segurança.

Trem 1	Sentido	Trem 2	Sentido	Tipo	Resultado
cv_1	direto	cv_3	direto	Unsafe	Não alcançada
cv_2	direto	cv_4	inverso	Unsafe	Não alcançada
cv_3	direto	cv_4, cv_5	qualquer	Unsafe	Não alcançada
cv_3	direto	cv_3	inverso	Unsafe	Não alcançada
cv_5	qualquer	cv_3, cv_4	direto	Unsafe	Não alcançada
cv_1	direto	cv_3	inverso	Safe	Alcançada
cv_5	qualquer	cv_4	do cv_2	Safe	Alcançada
cv_3	direto	cv_2	direto	Safe	Alcançada
cv_3	inverso	cv_2, cv_5	qualquer	Safe	Alcançada

Tabela 1: Posição relativa dos trens

Quanto à posição dos trens em relação aos AMVs, as situações inseguras na operação do sistema são: (i) o trem sai do cv_1, ocupa o cv_3 e se aproxima do AMV_3 que não está em *reverso*; (ii) o trem vem do cv_4, ocupa o cv_3 e se aproxima do AMV_3 quando este não está em *normal*; (iii) o trem vem do cv_3, ocupa o cv_4 e está próximo do AMV_4, que não está em *normal*; (iv) o trem vindo do cv_5 em sentido inverso, ocupa o cv_4 e se aproxima AMV_4, que não está em *normal*; e (v) o trem ocupa o cv_4, vindo do cv_2, e se aproxima do AMV_4, que não está em *reverso*. O resultado da verificação é apresentado na Tabela 2. A primeira

coluna indica o AMV. A segunda coluna indica o estado do AMV quando da passagem do trem. A terceira coluna indica de onde vem o trem. A quarta coluna representa a distância percorrida pelo trem dentro do *cv*. A quinta coluna apresenta a indicação da ferramenta HyTech, para cada situação. A marca zero é o ponto onde o trem invade o *cv*, trafegando no sentido da esquerda para a direita. O AMV está localizado à 200 metros desse ponto. Para operar em segurança, foi estipulado que o AMV deve estar travado quando o trem chegar à 20 metros do AMV. Da esquerda para a direita, essa marca está à 180 metros do ponto zero e, no sentido inverso, está à 20 metros do ponto zero. Novamente, uma inspeção da tabela revela que a operação do sistema é segura, nessas condições.

AMV	Posição	Trem de	Distância (m)	Resultado
AMV_3	não Reverso	cv_1	180	Não alcançada
AMV_3	não Normal	cv_4	20	Não alcançada
AMV_4	não Reverso	cv_2	20	Não alcançada
AMV_4	não Normal	cv_3	180	Não alcançada
AMV_4	não Normal	cv_5	20	Não alcançada
AMV_3	não Reverso	cv_1	160	Alcançada
AMV_3	não Normal	cv_4	40	Alcançada
AMV_4	não Reverso	cv_2	40	Alcançada
AMV_4	não Normal	cv_3	160	Alcançada
AMV_4	não Normal	cv_5	40	Alcançada

Tabela 2: Posição entre os trens e os AMVs

Os resultados obtidos a partir das análises realizadas são classificados em duas categorias. A primeira categoria estuda propriedades de segurança relacionadas à posição dos trens na via. A segunda categoria estuda o posicionamento seguro dos AMVs quando da passagem dos trens. A Tabela 3 mostra algumas características com relação ao tempo total de execução e consumo de memória, para algumas das análises realizadas, de uma forma geral. Cada linha da tabela, ilustra uma situação típica encontrada durante os exercícios de análise de propriedades do modelo. A primeira coluna identifica se a propriedade se refere à uma relação de posicionamento entre trens ou à uma relação de posicionamento entre um trem e um AMV, quando o trem se encontra no circuito de via onde se localiza o AMV. A segunda coluna destaca como a propriedade foi caracterizada para análise. Uma caracterização do tipo *unsafe* define predicados inseguros que descrevem regiões de configurações indesejadas para a operação do sistema. Já uma caracterização do tipo *safe* define predicados seguros, que mapeiam regiões de configurações seguras, ou seja, regiões que podem ser atingidas sem problemas para a operação segura do sistema. A terceira coluna apresenta o tipo de análise que foi utilizada para a verificação da propriedade. Ou seja, se foi usado o operador *Post*, para análise *forward*, ou o operador *Pre*, para análise *backward*. A quarta coluna mostra o número de iterações necessárias para que a análise

convirja. Algumas dessas análises não foram bem sucedidas, sendo interrompidas por falta de recursos de máquina ou da ferramenta. A quinta coluna apresenta o intervalo aproximado da quantidade de memória que foi utilizada para se completar as análises. Em algumas análises a memória total disponível não foi suficiente para a finalização da análise. A sexta e última coluna identifica o intervalo aproximado do tempo total gasto nas análises efetuadas. Por exemplo, a linha 5 da Tabela 3 informa que um dos exercícios de verificação da distância segura entre trens e AMVs foi modelado como uma propriedade do tipo *unsafe* e a análise evoluiu de forma *backward*, usando o operador *Pre*. Nessa execução, a ferramenta usou aproximadamente 300 segundos, necessitou de 7 iterações para convergir, e usou cerca de 500 MB, entre memória RAM e memória para *swap*.

Posicionamento	Caracterização	Análise	Iterações	Memória (MB)	Tempo (s)
Trens	Unsafe	Backward	2 a 3	350 a 450	110 a 260
Trens	Safe	Backward	–	–	–
Trens	Unsafe	Forward	–	–	–
Trens	Safe	Forward	61	$\geq$ 550	250 a 330
AMVs	Unsafe	Backward	7	450 a 550	280 a 300
AMVs	Safe	Backward	–	–	–
AMVs	Unsafe	Forward	82	200 a 250	150 a 160
AMVs	Safe	Forward	82	200 a 250	150 a 170

Tabela 3: Características gerais de algumas verificações

5 Conclusões

A operação de uma malha metroviária é crítica, exigindo uma validação rigorosa. Os estudos realizados nesse trabalho são um passo nessa direção. Como modelo formal foram usados autômatos híbridos. Esse formalismo lida bem com sistemas que apresentam perfis dinâmicos contínuos e eventos discretos, e onde há muitos componentes distribuídos. O modelo formal garantiu a segurança da operação de uma parte da malha metroviária, verificando aspectos do controle da movimentação de trens e do funcionamento de máquinas de chave. Modelos similares podem ser desenvolvidos para verificar outros aspectos do sistema. Como suporte computacional, foi usada a ferramenta HyTech, capaz de verificar autômatos híbridos lineares.

Referências

[1] Rajeev Alur, Thomas A. Henzinger, and Pei-Hsin Ho. Automatic Symbolic Verification of Embedded Systems. In *Proceedings of the 14th Annual IEEE Real-Time Systems*

Symposium, pages 2–11. IEEE Computer Society Press, 1993.

[2] J. Bengtsson, K. G. Larsen, F. Larsson, P. Pettersson, and Wang Yi. UPPAAL in 1995. *Lecture Notes in Computer Science*, 1055:111–114, 1996.

[3] Johan Bengtsson, Kim Larsen, Fredrik Larsson, Paul Pettersson, Wang Yi, and Carsten Weise. New Generation UPPAAL. Technical report, BRICS, Dept. of Computer Science, Aalborg University, Denmark and Department of Computer Systems, Uppsala University, Sweden. Esse relatório técnico pode ser encontrado no endereço http://www.docs.uu.se/docs/rtmv/uppaal.

[4] Johan Bengtsson, Kim G. Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. UPPAAL — a Tool Suite for Automatic Verification of Real–Time Systems. Technical Report RS-96-58, BRICS, Aalborg University, DENMARK and Department of Computer Systems, Uppsala University, Sweden, December 1996.

[5] Adilson Luiz Bonifácio and Itana Maria de Souza Gimenes. Estudo e comparação de provadores automáticos de teoremas. *Revista Tecnológica*, (7):75–85, Outubro 1998. Departamento de Ciência da Computação, Universidade Estadual de Maringá, Brasil.

[6] Adilson Luiz Bonifácio, Arnaldo Vieira Moura, João Batista Camargo Jr., and Jorge Rady Almeida Junior. Análise, Verificação e Síntese de Segmentos de Via de uma Malha Metroviária. Technical Report 99-18, Instituto de Computação, Universidade de Campinas, Agosto 1999.

[7] José Henrique Zaccardi de Freitas, Antonio Accurso, and Ivaldo Lopes Mathias. A Evolução Tecnológica da CMSP e o Estado da Arte de Sistemas de Sinalização Baseado em Comunicação. In *Revista Engenharia*, volume 529, pages 116–124, 1998.

[8] Thomas A. Henzinger and Pei-Hsin Ho. HyTech: The Cornell Hybrid TECHnology Tool. *Workshop on Hybrid Systems and Autonomous Control*, October 1994.

[9] Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. HYTECH: the next generation. In *Proceedings of the 16th IEEE Real-Time Systems Symposium*, pages 56–65, Pisa, Italy, 5-7, December 1995. IEEE Computer Society Press.

[10] Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. A user guide to HyTech. In E. Brinksma, W.R. Cleaveland, K.G. Larsen, T. Margaria, and B. Steffen, editors, *TACAS 95: Proceedings of the First Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, volume 1019 of *Lecture Notes in Computer Science*, pages 41–71. Springer-Verlag, 1995.

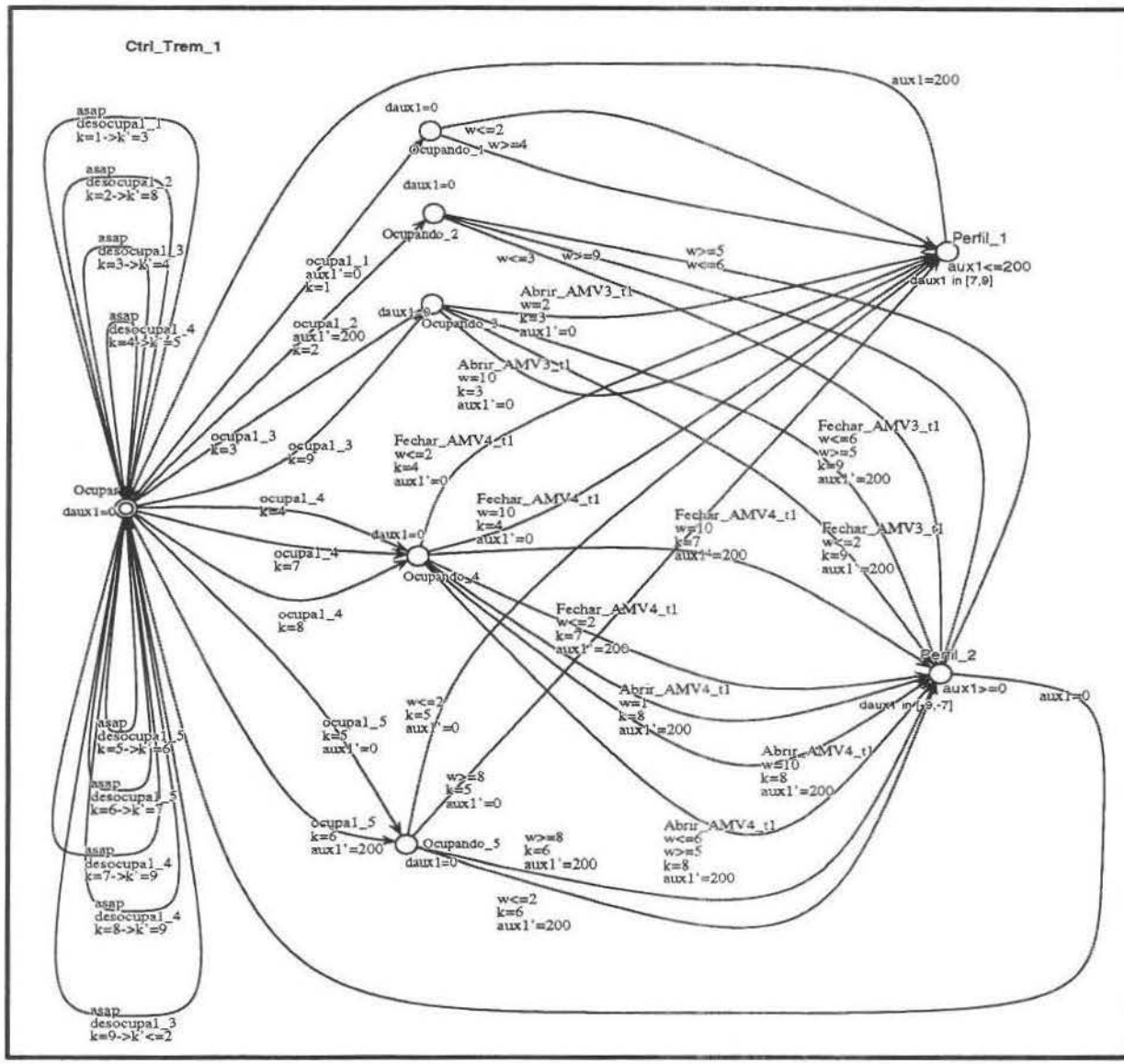
[11] Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. HYTECH: A model checker for hybrid systems. In O. Grumberg, editor, *CAV'97: Proc. of the Ninth Int. Conference on Computer-Aided Verification*, volume 1254 of *Lecture Notes in Computer Science*, pages 460–463. Springer-Verlag, 1997.

[12] Thomas A. Henzinger and Howard Wong-Toi. Using HyTech to Synthesize Control Parameters for a Steam Boiler. In J.-R. Abrial, E. Börger, and H. Langmaack, editors, *Formal Methods for Industrial Applications: Specifying and Programming the Steam Boiler Control*, volume 1165 of *Lecture Notes in Control and Information Science*, pages 265–282. Springer-Verlag, 1996.

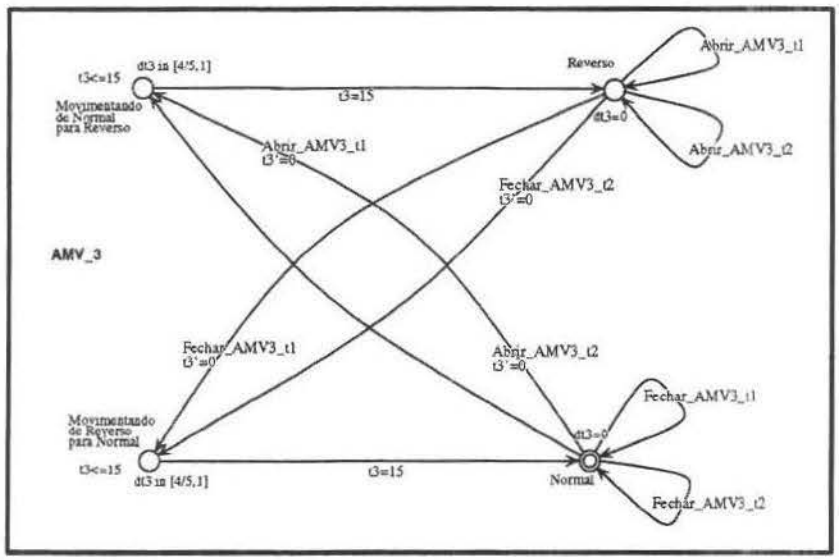
[13] Pei-Hsin Ho. *Automatic Analysis of Hybrid Systems.* PhD thesis, Cornell University, August 1995.

[14] Pei-Hsin Ho and Howard Wong-Toi. Automated analysis of an audio control protocol. In P. Wolper, editor, *Proceedings of the 7th International Conference On Computer Aided Verification*, volume 939 of *Lecture Notes in Computer Science*, pages 381–394, Liege, Belgium, July 1995. Springer-Verlag.

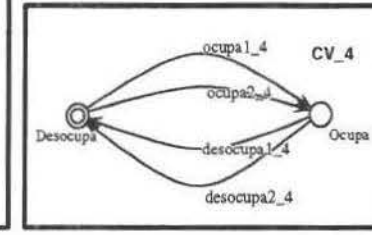
[15] Kim G. Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a NUTSHELL. *International Journal on Software Tools for Technology Transfer*, 1(1):134–152, December 1997.



(a) Autômato do trem



(b) Autômato do AMV



(c) Autômato da via

Figura 3: Os autômatos do modelo

Epílogo

Este capítulo apresentou um artigo que demonstra a verificação de aspectos de segurança da operação de trens trafegando numa malha metroviária. Através da verificação de diversas propriedades, foi possível detectar sob quais condições o sistema se mantém seguro.

O artigo enfocou vários aspectos e funcionalidades do sistema metroviário, com o objetivo central de evitar colisões e descarrilhamentos. Aspectos de controle dos trens e das máquinas de chave foram modelados usando a abordagem formal de autômatos híbridos. Tais autômatos permitem a especificação dos componentes mecânicos, referente a parte analógica do sistema, e também permitem a especificação de controles discretos, referentes a parte de comunicação do sistema.

Devido a complexidade da malha, apenas um segmento de via foi modelado, aquele localizado no fim da via e que é usado como pátio de manobras para retorno dos trens.

Capítulo 5

Síntese de Parâmetros para Segmentos de uma Malha Metroviária

Prólogo

O artigo denominado "Formal Parameters Synthesis for Track Segments of a Subway Mesh", apresentado neste capítulo, focaliza o mesmo segmento de malha estudado no artigo apresentado no Capítulo anterior.

No presente artigo, porém, o objetivo central é a síntese de parâmetros críticos para a operação segura do sistema. Vários valores numéricos são substituídos por constantes simbólicas. A ferramenta HyTech, então, sintetiza intervalos para estes valores, de forma a ainda manter o sistema operando em segurança. Os intervalos sintetizados possibilitam a escolha de valores mais justos para certos parâmetros críticos da operação do sistema, sem que sua segurança seja prejudicada. A economia obtida pode ser tanto de recursos materiais, como custos de operacionalização, disponibilidade do sistema, ou até mesmo de tempo na operação da malha.

Este artigo foi submetido e aceito para publicação nos anais da *7th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems* (ECBS'00), a ser realizado de 3 a 7 de abril de 2000 em Edinburgh, na Escócia. Os anais serão editados pelo *Institute of Electrical and Electronic Engineers* (IEEE).

Formal Parameters Synthesis for Track Segments of a Subway Mesh

Adilson L. Bonifácio Arnaldo V. Moura
Computing Institute, Univ. of Campinas
P.O. 6176, Campinas, Brazil, 13081-970
Support: CAPES, DS - 44/97, FAPESP, 98/05999-4
{albonifa,arnaldo}@dcc.unicamp.br

João B. Camargo Jr. Jorge Rady de A. Jr.
Politechnic School, Univ. of São Paulo
Av. Prof. L. Gualberto, Trav. 3, N. 158,
São Paulo, Brazil, 05508-900
{jbcjunio,jrady}@pcs.usp.br

Abstract

The aim of this work is to apply formal specification techniques to model real-time distributed systems arising from real-world applications. The formal models discussed here are based on the notion of hybrid automata. The target system is the maneuvering yard of a subway mesh. Semi-automatic tools are used in the analysis and verification of the models here developed. The models are also used to synthesize some important parameters of the system under consideration. All results were obtained on a typical 350MHz desktop PC, with 320MB of main memory.

1 Introduction

The use of formal specifications in the system development process has become more and more important, especially when such processes focus on distributed systems that control critical applications, where an operational fault can cause irreparable damages. Among a wide variety of distributed systems, the ones involving reactive and real-time responses are the most difficult to model accurately. In addition, certain kind of critical applications, known as *hybrid systems*, present a dual nature, in the sense that the system behavior is described by continuous dynamic profiles, regulated by the intervention of discrete events.

Recently, the mathematical theory of hybrid automata has been gaining momentum as a good alternative to specify and verify hybrid systems [1, 10, 12, 14]. Essentially, hybrid automata are finite automata whose states include a dynamic specification of the system evolution, and whose transitions model an abrupt change in the dynamic behavior of the system. Each component of the distributed system is modeled by an appropriate automaton. The overall system behavior is, then, captured by the corresponding product

automaton [12, 14]. The communication between the system components is regulated by the exchange of messages between the independent automata. Messages are exchanged during transitions taking place in the individual automata. Synchronism between the individual automata is also attained by the use of a shared memory, to which all the component automata have access [12, 14]. Usually, classic control theories deal with a few distributed components whose dynamic features are complex. The notion of hybrid automata focuses on several distributed communicating components with a simpler dynamic behavior.

In this work, hybrid automata are used to model the complexities of a real-world distributed hybrid system, consisting of a maneuvering yard located at the end of the tracks, in a larger subway mesh. The yard, used to maneuver returning trains, is one of the critical segments of the mesh. Although of a relatively simple topology, the subway system encompasses a large number of subcomponents and sensors, spread over all the extension of the mesh. Moreover, its operation requires an intense exchange of messages and a complex activity of coordination between these physical components. In order to manage this complexity, the global system was divided into simpler segments which could be more easily modeled, and in such a way that the semi-automatic analysis tools could, effectively, be brought to bear on the simpler models. The results of the analysis on the several segments contribute a step towards the validation and the verification of the global system properties. The authors are unaware of similar applications of hybrid automata to the problem of synthesizing operational parameters of a real subway mesh.

Most of the time, models for realistic systems result in a product automaton of such complexity that the cooperation of automatic tools becomes essential. Here, the HyTech software tool [9, 11, 10] was used to analyze the models.

Section 2 presents the notion of hybrid automata. Section 3 describes the subway mesh. Section 4 discusses the models that were developed and describes the synthesis of values for certain important system parameters, in such a way that the overall system safety remains guaranteed. The last section concludes with some final observations.

2 Hybrid Automata

A *hybrid automaton* is a formal system given by $A = (X, V, \mathit{flow}, \mathit{init}, \mathit{inv}, E, \mathit{jump}, \Sigma, \mathit{syn})$, where:

1. $X = \{x_1, x_2, \ldots, x_n\}$ is a finite set of *variables*. The number n is called the *dimension* of A.

2. V is a finite set of *operation modes*.

3. For every mode $v \in V$, *flow(v)*, the *continuous activity condition* of mode v, is a predicate over the set of variables $X \cup \dot{X}$. Here, $\dot{X} = \{\dot{x}_1, \ldots, \dot{x}_n\}$.[1]

4. For every mode $v \in V$, *inv(v)*, the *invariant condition* for mode v, is a predicate over X. When in mode v, the variables must satisfy the invariant condition *inv(v)*.

5. The *initial conditions* are given by the component *init*. For every mode $v \in V$, *init(v)* is a predicate over X. The automaton can start in mode v only if the initial condition *init(v)* is satisfied.

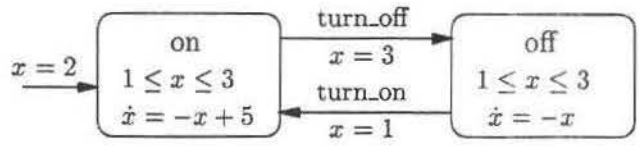
6. The multi-set E is formed by pairs (v, v'), where $v, v' \in V$ are operation modes. The multi-set describes the *transitions* of the automaton.

7. The component *jump* describes the *phase change conditions*. For every transition $e \in E$, *jump(e)* is a predicate over the set $X \cup X'$. Here, $X' = \{x'_1, \ldots, x'_n\}$. The primed variable x' is used to indicate the (new) value of the corresponding variable x after a transition.

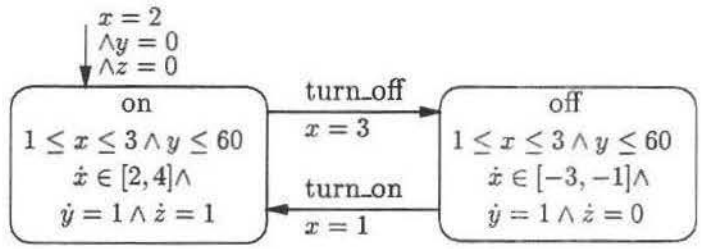
8. Σ is a finite set of *events labels*. The partial function *syn* maps transitions in E into events of Σ.

A simple and classical example [12] of a hybrid automaton is shown in Figure 1(a). This model uses real variable x, which evolves deterministically in time, to model the temperature of the container. The heater is turned off when the temperature reaches 3 degrees, and turned back on when the temperature falls to 1 degree. The evolution of the real variables is computed from the initial conditions, from differential equations present in the modes, and form the transitions between modes. In the example, it is clear that $n = 1$, $X = \{x\}$, and that $V = \{on, off\}$. The continuous activity condition for mode *on* is given by the predicate $\dot{x} = -x + 5$, which describes an exponential rise of the temperature x. For mode *off* the continuous activity condition is $\dot{x} = -x$, describing an exponential fall in the temperature. For both operation modes, the invariant condition is $1 \leq x \leq 3$. The initial condition for mode *on* is $x = 2$, and for mode *off* it is `false`. The latter condition is not depicted in the figure. There are two transitions in Figure 1(a), namely (*on*, *off*) and (*off*, *on*). The transition (*on*, *off*) has a phase change condition given by the predicate $x = 3 \wedge x' = x$, and the transition (*off*, *on*) has the phase change condition given by the predicate $x = 1 \wedge x' = x$. The trivial condition $x' = x$ is not shown in the figure. In this case, the value of the real variables do not change when a transition is taken. The set of discrete events is $\Sigma = \{turn\_on, turn\_off\}$. The function *syn* associates the first event to the transition (*off*, *on*), and associates the second event to the transition (*on*, *off*).

[1] For any variable x, the first derivative of x with respect to time is indicated by $\dot{x}$.



(a) Simple heater



(b) Heater and safety requirement

Figure 1: Hybrid automata: an example

The presence of these events allows the synchronization between distributed automata, as described in the sequel.

Given the internal requirements for mode *on*, in the example depicted in Figure 1(a), it is easy to verify that the predicate $2 \leq \dot{x} \leq 4$ is always satisfied. Similarly, it is easy to see that the condition $\dot{x} \in [-3, -1]$ is always satisfied in mode *off*. Using these facts, a non-deterministic relaxation of the previous model can be defined, as shown in Figure 1(b). The latter model also uses two extra variables, y and z. Variable y is a perfect clock, its time derivative being one. Variable z is a perfect stopwatch, since its time derivative is either zero (clock stopped) or one (clock running). These two extra variables can be used to describe external safety conditions, such as "during the first hour of operation, the heater shall not be on for more than 40 minutes" [12]. It is common to use "guards", in the sense of [7], to represent the phase change conditions in the graphical representation of hybrid automata. A guard such as $(x_1 = x_2) \rightarrow (x_1 := 2x_2)$ gives the precondition, $x_1 = x_2$, and the postcondition, $x_1' = 2x_2$, that are to be enforced during a phase change. These conditions are equivalent to the complete predicate $x_1 = x_2 \wedge x_1' = 2x_2 \wedge x_2' = x_2$.

The model of a complex system comprises several individual automata that operate concurrently. The synchronism of the global system is obtained by:

1. imposing synchronous phase transitions that are labeled by the same discrete event,

and

2. using shared variables.

Let A_1 and A_2 are hybrid automata, where $X_1 \cap X_2 = \emptyset$. The *product automaton* of A_1 and A_2 is a system $A = (X_1 \cup X_2, V_1 \times V_2, \mathit{flow}, \mathit{init}, \mathit{inv}, E, \mathit{jump}, \Sigma_1 \cup \Sigma_2, \mathit{syn})$, where the *flow*, *init* and *inv* components are simple conjunctions: $\mathit{flow}((v_1, v_2)) = \mathit{flow}(v_1) \wedge \mathit{flow}(v_2)$; $\mathit{inv}((v_1, v_2)) = \mathit{inv}(v_1) \wedge \mathit{inv}(v_2)$; $\mathit{init}((v_1, v_2)) = \mathit{init}(v_1) \wedge \mathit{init}(v_2)$. For the phase transitions, $e = ((v_1, v_1'), (v_2, v_2')) \in E$ if and only if one of the following conditions hold:

1. $v_1 = v_1'$, $e_2 = (v_2, v_2') \in E_2$ and $\mathit{syn}_2(e_2) \notin \Sigma_1$; $\mathit{jump}(e) = \mathit{jump}_2(e_2)$ and $\mathit{syn}(e) = \mathit{syn}_2(e_2)$.

2. $v_2 = v_2'$, $e_1 = (v_1, v_1') \in E_1$ and $\mathit{syn}_1(e_1) \notin \Sigma_2$; $\mathit{jump}(e) = \mathit{jump}_1(e_1)$ and $\mathit{syn}(e) = \mathit{syn}_1(e_1)$.

3. $e_1 = (v_1, v_1') \in E_1$, $e_2 = (v_2, v_2') \in E_2$, $\mathit{syn}_1(e_1) = \mathit{syn}_2(e_2)$. The synchronism is imposed by requiring that $\mathit{jump}(e) = \mathit{jump}_1(e_1) \wedge \mathit{jump}_2(e_2)$ and that $\mathit{syn}(e) = \mathit{syn}_1(e_1)$.

The product of several automata is obtained by accumulating the result of computing the product of each individual automaton in turn.

An *atomic linear predicate* is an inequality between rational constants and linear combinations of variables with rational coefficients, such as $2x + 4y - 7z/2 \leq -10$. A *convex linear predicate* is a finite conjunction of atomic linear predicates, and a *linear predicate* is a finite disjunction of convex linear predicates. A hybrid automaton A is a linear hybrid automaton, if it satisfies the following requirements: (i) For every operation mode $v \in V$, the continuous activity condition *flow(v)*, the invariant condition *inv(v)* and the initial condition *init(v)*, are convex linear predicates. Moreover, for every transition $e \in E$, the phase change condition *jump(e)* is a convex linear predicate; (ii) For every operation mode $v \in V$, the continuous activity condition *flow(v)* is a predicate over the set $\dot{X}$. The second requirement ensures that the dynamic profile of the continuous variables depends only on the operation mode, and does not depend on the value of these variables. This condition prohibits continuous activities such as $\dot{x} = x$. On the other hand, it still allows the use of stopwatches and clocks with bounded drift, the latter being described by dynamic conditions such as $\dot{x} \in [1 - \epsilon_1, 1 + \epsilon_2]$, for some constants ϵ_1 and ϵ_2.

A *convex region* of a hybrid automaton A of dimension n is a convex polyhedron in $\mathbb{R}^n$. A *region* is a finite union of convex regions. Given a predicate φ, the region determined by φ is denoted by $[\varphi]$, and is called the φ-region. A *configuration* of a hybrid automaton A is a pair $q = (v, a)$ consisting of an operation mode $v \in V$ and a vector $a = (a_1, \ldots, a_n)$ in $\mathbb{R}^n$. The configuration (v, a) of A is *admissible* if $a \in [\mathit{inv}(v)]$. The configuration (v, a) is

initial $a \in [init(v)]$. Note that the *inv* region is formed by all the admissible configurations and, in the same vein, the *init* region is characterized by all the initial configurations.

Let $q = (v, a)$ and $q' = (v', a')$ be two configurations of A. The pair (q, q') is a *phase change* of A if $e = (v, v')$ is a transition in E and $(a, a') \in [jump(e)]$. The pair (q, q') is a *continuous activity* of A if $v = v'$, if there is a nonnegative real $\delta \in \mathbb{R}$ (the duration of the continuous activity) and if there is also a differentiable function $\rho : [0, \delta] \to \mathbb{R}^n$ (the curve of the continuous activity), such that the following three requirements are satisfied: (i) Endpoints: $\rho(0) = a$ and $\rho(\delta) = a'$. (ii) Invariant condition: for all time instants $t \in [0, \delta]$, the configuration $(v, \rho(t))$ is admissible; (iii) Continuous activity condition: let $\dot{\rho} : [0, \delta] \to \mathbb{R}^n$ be the first derivative of ρ. For all time instants $t \in [0, \delta]$, $\rho(t) \in [inv(v)]$ and $(\rho(t), \dot{\rho}(t)) \in [flow(v)]$.

A *trajectory* of a A is a finite sequence $q_0, q_1, \ldots, q_k$, of admissible configurations, where each pair (q_j, q_{j+1}) of consecutive configurations is either a phase change or a continuous activity of A. Such a trajectory is said to *start* at q_0. A configuration q' of A is *reachable from* a configuration q if q' is the last configuration of some trajectory of A starting at q. An *initial trajectory* of A is any trajectory of A starting at some initial configuration of A. A configuration q' of A is simply *reachable* if it is reachable from a initial configuration of A. Given a region φ, $Post(\varphi)$ is the region formed by all those configurations q' for which there exists a configuration $q \in [\varphi]$ such that q' is reachable from q through a continuous activity or of a phase change of A. Starting with $\varphi_0 = \varphi$, the iteration of this process will compute the regions $\varphi_{k+1} = Post(\varphi_k)$, for $k = 0, 1, \ldots$. If a region φ_k satisfies $\varphi_k = \varphi_{k+1}$, then the process converges and region φ_k contains all the configurations of A that are reachable via some trajectory that starts from a configuration in φ_0. When the process converges, region φ_k is denoted by $Post^*(\varphi_0)$. A backward process can also be defined in a similar way, giving rise to a *Pre* operator. The following theorem can be shown [1]: if A is a linear hybrid automaton, and if $[\varphi]$ is any region of A, then the calculation of $[\varphi]$ converges. This theorem supports the construction of (semi)-automatic tools for the analysis of linear hybrid automata.

A *safety requirement* is a set of restrictions and conditions imposed on the system configurations. A configuration is *safe* if it satisfies all the safety conditions associated with the model; otherwise the configuration is *unsafe*. A model is *safe* if all its reachable configurations are safe; otherwise the model is *unsafe*. A safety requirement is described by the set of values that the system variables can attain. Given a safety requirement as a predicate φ over the configurations of A, the region $reach = Post^*([init])$ is computed, and the intersection $reach \cap [\neg\varphi]$ is obtained. If the resulting region is empty, the safety requirements are satisfied; otherwise they are violated.

System parameters are symbolic constants which assume fixed values. A parametric analysis determines value intervals for certain system parameters in such a way as to guar-

antee that no safety requirement is violated. This process, therefore, synthesizes maximum and minimum values for these parameters. In a hybrid automaton, a parameter α can be represented by a variable whose value never changes. This condition can be attained by imposing the condition $\dot{\alpha} = 0$ in all operation modes and, further, by imposing the condition $\alpha' = \alpha$ over all transitions of the automaton. A value $a \in \mathbb{R}$ is *safe for a parameter* α if no unsafe configuration is reachable when the initial condition $\alpha = a$ is adjoined to the all operating modes of the automaton.

In the example depicted on Figure 1(a), the configuration $(on, 0.5)$ is not admissible, but all admissible configurations are reachable. In the same example, it is easy to see that the configuration $(on, 2)$ is initial. A safety requirement for the heater could be: "in the first hour of operation, the heater should not be on by more than 2/3 of the time." Figure 1(b) depicts a relaxed version of that automaton, where the new variable y measures the total elapsed time and the new variable z measures the time the heater is in operation. It can be demonstrated that the relaxed version is safe with respect to these requirements [12]. It follows that the original version is also safe with respect to the same requirements.

In general, the modeling of realistic systems results in a product automaton of such complexity that the verification phase can only be successfully done with the aid of (semi)-automatic computational tools. The software HyTech [10, 12, 11, 9] was developed as a tool to aid in the analysis and verification of linear hybrid automaton models. The model is described via an input file that comprises two parts. The first part describes the individual hybrid automata. The product automaton is computed automatically by the tool. The second part is a sequence of analysis commands. The HyTech tool can automatically verify that the relaxed heater model illustrated in the Figure 1(b) satisfies all the posed safety requirements [12]. Nowadays, the tool is being used in a variety of diverse situations and further examples of its use can be found in [1, 12, 11, 9]. Today, HyTech is one of the only tools that supports (semi)-automatic analysis of linear hybrid automata. Other tools, as the UPPAAL [16, 2, 3, 4], are based on timed-automata, a notion more restricted than that of linear hybrid automata. The former can only work with clocks, and does not support the use of more complex differential equations to describe system dynamic profiles. A number of other related works, dealing with hybrid systems, can be found in [15, 13, 14].

3 Description of the Subway Mesh Segment

Subway systems are complex distributed systems that function under tight operational conditions. Needless to say, any safety fault can result in intolerably high damages, both in terms of properties and lives. Hence, the need arises for a formal, rigorous, verification of all the system safety requirements. Part of the subway traffic is controlled using signaling electronic systems [8], implemented on microprocessors. Recent technological advances in

the area of micro-processed circuits make possible the use of more sophisticated functions in the control and supervision of the system behavior. This more advanced technology permits a reduction of the interval between consecutive trains, thereby increasing the system capacity, while still maintaining adequate levels of security, promptness and reliability, and within an adequate cost. The use of more advanced technologies tightens further the system parameters and accrues the verification and safety problems, opening up the possibilities for introducing subtle errors. In these cases, a formal verification of the system safety is the more desirable.

The emphasis of this work is on the analysis and verification of safety aspects of a subway mesh. The aim is, on the one hand, to model the system and to verify different properties of the model, validating its operation. On the other hand, the work here described also focuses on the synthesis of more appropriate and tighter values for some critical system parameters, in such a way as to improve the system overall performance. One of the main aspects of the modeling is to capture how and when the cooperation between the system parts takes place.

The functioning and the operation of the relevant essential features of a subway mesh that are necessary to understand the modeling phase are described next:

1. Track Circuits (*tcs*): these are segments where a train can be detected. In a track circuit, speed codes are transmitted to the trains. The enforcement of such codes guarantees the safety of the trains, given their position in the track, and in accordance with their relative position with respect to other trains.

2. Switch Machines (*SM*s): switch machines are located on the tracks and permit a train to change tracks. The machines must be correctly positioned and locked, each time a train passed through. They are operated electrically (or manually in emergency conditions), and can be in one of two states: (i) normal; and (ii) reverse. In each state, the machine can be either electrically locked or electrically unlocked.

3. Terminal Zones: it is a yard comprising several track circuits, which are used for maneuvering.

4. Interlock Regions: these are segments where the *SM*s are found. They are also called "*SM*s regions".

5. Micro-processed Interlock: this equipment communicates with the *SM*s and the track circuits. It is part of the automatic control of the global system, and it automatically supervises the safe movement of the trains.

Some functions of the subsystem that controls the movement of the trains are: (1) selection of the interlock control mode; (2) issue commands to the *SM*s; (3) impose speed

restrictions; and (4) issue commands to reverse the traffic direction. Reversing the traffic direction allows a train to switch traffic direction at a terminal zone and also allows for the rescue of blocked trains.

The basic safety requirements are to prevent collisions and derails. Some safety conditions are: (a) occupy a *tc* only if there is no traffic in the opposite direction; (b) occupy a *tc* only if it is empty; (c) indicate the release of a *tc* when the train abandons it; (d) unlock a *SM* only when the corresponding *tc* is empty; (e) enforce the speed profiles; (f) reverse the traffic direction only when the *tc* is empty. More details are given in [6].

A map of a terminal track section is depicted in Figure 2, and it includes five *tc*s, and two *SM*s. Each *tc* is about 200 meters in length. The occupation of a *tc* is provoked by a

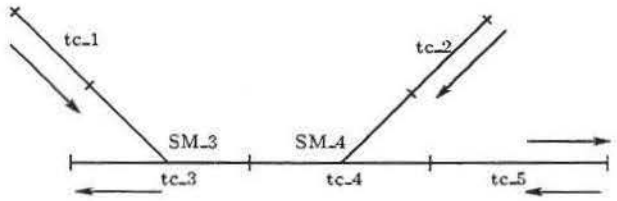


Figure 2: Tracks in a maneuvering yard

request from some train that needs to pass though the *tc* in order to arrive at its destination. When the train leaves the *tc*, the latter is released and it is, thus, ready to be occupied again by some train traveling on the same tracks. If the occupation is granted, adequate speed profiles are then imposed using suitable transmitters attached to the *tc*. The basic speed profiles for the system are: 0 km/h, 30 km/h, 60 km/h, 80km/h and 100 km/h. There is a predefined sequence of default speed profiles defined along each track. During the system operation, the speed code of 0 km/h is applied to any circuit that cannot be occupied. Also, a circuit that contains a *SM* carries a standard speed profile of 30 km/h, and track circuits farther away from interlock regions can have speed profiles of up to 100 km/h. Moreover, the occupation of any particular *tc* forces a particular speed profile on the *tc*s that fall behind it. These *tc*s have a speed limit imposed upon them in agreement to their distances to the base *tc*. This scheme, called a *shadow*, is a safety condition imposed on the system operation to avert collisions.

When a train occupies a *tc* that contains a *SM*, the train must have, previously, requested the movement of the *SM* to the required position, either to *normal* or to *reverse*. After the train signals the positioning of the *SM*, the latter unlocks and starts to move to the required position. This takes some time, until the needle point locks in the correct place and the engine subsequently disengages. Enough time has to be allowed for this movement to complete, before the train is about to cross the *tc* where the *SM* is located.

4 Parameter Synthesis

For safety reasons, many subway traffic control systems require the positioning and locking of any *SM* before the train invades the *tc* where the *SM* is located. The model treated in this work deviates from this notion. Here, the system dynamics allows a train to enter the *tc* before the *SM* is locked. Experiments of this kind can reveal the limits of a safe system operation and can also be instrumental in obtaining tighter values for a number of system parameters.

The system modeled in this work comprises the terminal track yard depicted in Figure 2, plus two trains traveling from left to right on the upper tracks, and reversing direction on the lower tracks. The full system model comprises nine individual hybrid automata: one for each train; one for each *SM*; and five other automata, one for each one of the five *tc*s shown. The two train automata are the more complex of the nine automata. Each one has eight states that represent the various situations that arise when the train passes through the yard. One of the train model is shown in Figure 10. Variable $aux1$ indicates the distance of the trains inside of the currently occupied *tc*, counting from the *tc* entrance point. Variables k and w are used to enforce the occupation rules. The automaton starts off in the "Get" state. From this state, it moves to one of the "Taking"states, depending on the value of variable k. This variable identifies which of the *tc*s is being occupied next. The two "Profile" states distinguish between the two speed codes imposed on the two *SM* regions. A positive speed indicates movement from left to right, and a negative one signals movement in the opposite direction. When a *tc* is released, the "Get" state is reentered, and the cycle repeats.

Referring back to Figure 2, at the start of the cycle the train is about to reach the maneuvering yard, traveling from left to right on the upper tracks. At this point, it non-deterministically takes either tc_1 or tc_2, and starts to maneuver towards the lower tracks. If tc_3 is free, the train might choose to go down tc_1. After it enters tc_1, the train starts to travel at a speed that varies from 7 to 9 meters per second, until it crosses the 200 meter extension of tc_1, and is ready to occupy tc_3. If the latter is occupied, the train waits until it is released. Also, the train only occupies tc_3 if tc_4 and tc_5 are both free. Upon occupying tc_3, the train signals that it needs the positioning of SM_3 in the *reverse* state. Once that action is completed, the train crosses SM_3 and continues on tc_3. Before leaving tc_3, the train requests the occupation of tc_4. Upon being granted access to tc_4, the train signals for the movement of SM_4 to the *normal* state, and goes on to complete the crossing of tc_4. When reaching tc_5 the train travels all its extension and, at its end, the train reverses direction. It, then, starts traveling from right to left with a speed between -9 to -7 meters per second. Upon reaching the end of tc_5, the train will only occupy tc_4 if it is already free. When entering tc_4, the train signals for the positioning of SM_4 in the *normal* state,

proceeds to cross the SM_4 sector, and travels all the way towards the end of tc_4. Next, it occupies tc_3 and signals for the movement of SM_3 to the *normal* state. After traveling all the extension of tc_3, the train abandons this circuit and returns to the initial point.

Alternatively, from the initial position, the train might non-deterministically decide to travel down tc_2. In this case, it will only occupy tc_4 if there is no train in tc_3 traveling in the opposite direction, nor there is another train occupying tc_4. When it reaches tc_4, the train signals for the movement of SM_4 to the *reverse* state. When that movement is completed, the train continues towards the left end of tc_4. Having reached it, the train frees tc_4 and enters tc_3, signaling for the movement of SM_3 to the *normal* position. After the train crosses tc_3, it returns to the initial position and restarts the cycle.

The *SM* automaton is formed by four states as shown in Figure 11. Variables $t3$ and $t4$ are used to time the movement of the *SM*s. The automaton is initiated in the *normal* state. The time it takes to complete a move, in either direction, is 15 seconds. There is a tolerance in this value. This imprecision is modeled by nondeterministically choosing a clock rate in the interval $[0.8, 1]$. The discrete events "Open"and "Close" synchronize the train and the *tc* automata. The model of a *tc* is illustrated in Figure 12. Each one has only two states, namely, "Release" and "Take". The product of all the automata forms the product automaton that models the system.

All computational results were obtained by running the HyTech tool on a typical 350MHz desktop Pentium II PC, with 320MB of RAM and 420MB of *swap* space. One of the major difficulties faced in the development phase was caused by excessive resource consumption when running the HyTech tool. This problem was more acute when the tool attempted to compute the product automaton. To overcome the memory usage problem, a careful *bottom-up* construction of the models was undertaken, always working at the tool limits. Even so, a number of compromise decisions had to be reached, in order to adjust the full model to the limits of the available hardware. The number of trains was limited to two, and only two speed profiles were allowed. In order to gain a perspective on the resource consumption during the experiments, Table 1 shows some typical data collected from these runs. The first column numbers the lines in the table. The second column indicates how the safety condition was characterized: by describing the safe region, or by giving the region of undesirable, unsafe, configurations. The third column indicates the nature of the iteration process that converged: a forward analysis used the *Post* operator, and a backward analysis used the *Pre* operator. The fourth column shows the number of iterations that were necessary to reach convergence. The fifth column presents the approximate amount of RAM and swap memory that was used to complete the analyses. The sixth column gives the approximate running time for the corresponding experiment to reach completion. Some of these experiments were aborted due to insufficient machine resources. For example, from line five one reads that in one of the experiments the safety condition

	Prop.	Anal.	Iterat.	Mem. (MB)	Time (s)
1	Uns.	Back.	2 to 3	350 to 450	110 to 260
2	Safe	Back.	–	–	–
3	Uns.	Forw.	–	–	–
4	Safe	Forw.	61	> 550	250 to 330
5	Uns.	Back.	7	450 to 550	280 to 300
6	Safe	Back.	–	–	–
7	Uns.	Forw.	82	200 to 250	150 to 160
8	Safe	Forw.	82	200 to 250	150 to 170

Table 1: Typical resource consumption

was modeled by describing the region of unsafe configurations, that the analysis proceeded backwards, that the run needed 7 iterations to converge, that the execution time for this experiment was approximately 300 seconds, and used about 500MB of RAM.

The rest of this section reports on parameters synthesis for critical values of the model. Other work [5, 6] reported on safety property verification for the same real system.

Due to machine resource restrictions, during synthesis, a simplification of the train automaton was used. Since all synthesis experiments were local to a particular *tc*, only one "Profile" and one "Taking" state were maintained.

4.1 The maximum distance inside a *tc*

Here, the focus is on the maximum distance that the train can travel, inside the *tc*, before signaling for the *SM* to move, while still guaranteeing all safety conditions. The results obtained appear in Table 2. The first column indicates the distance, inside the track circuit, of the point where the train requests the *SM* to start moving. The second column shows the maximum distance that the train can reach, inside the track circuit, up until the moment when the *SM* locks. In the limit, if this distance measures all the 200 meters of a track circuit, the train could still pass through the *SM*. The third column indicates the time for the *SM* to complete its movement. In all of these simulations, the train speed was assumed to be in the interval between 7 and 9 meters per second, as the fourth column of the table shows. From line 1 in Table 2, it can be seen that if the trains signal the *SM* to move immediately after entering the *tc*, then the maximum distance it can travel inside the *tc*, and before the *SM* locks, is 675/4 meters. Given that the *tc* total extension is 200 meters, it follows that the system operates safely if the trains always signal for the *SM* to move right upon entering the *tc*. Further, by consulting line 3 of Table 2, it can be seen that the largest mark on which the trains could signal the *SM* to move is the point located at 125/4 meters past the *tc* entrance, given that the *tc* extension is 800/4 meters.

Figure 3 illustrates the output of the HyTech tool for the experiment characterized by

Signal (m)	Max. dist. (m)	Time (s)	Speed (m/s)
0	675/4	15	[7,9]
20	755/4	15	[7,9]
125/4	800/4	15	[7,9]
40	835/4	15	[7,9]

Table 2: Max. dist. after signaling the *SM*

the first line in Table 2. The variable *train_dist* indicates the distance traveled by the

```
Number of iterations required for
reachability: 10
     train_dist >= 0 & 4train_dist <= 675
```

Figure 3: Param. analysis of latest point to signal

train after entering the track circuit. The output of the tool is the clause ($train\_dist \geq 0$) $\wedge$ ($train\_dist \leq 675/4$). Disregarding the trivial condition $train\_dist \geq 0$, the output clause reduces to $train\_dist \leq 675/4$. In this exercise, the unsafe *SM* region was given as a target region on input to the tool. Hence, the output clause produced indicates the condition under which the unsafe *SM* region can be reached. That is, the *SM* region is unsafe when the trains has traveled any distance from 0 to 675/4 meters. Since the *SM* is at a distance of $800/4 = 200$ meters from the entrance of the *tc*, one can conclude that the operation is safe, provided that the train signals the *SM* upon entering the corresponding *tc*.

Another study was performed in order to disclose what is the safe relationship between the maximum distance traveled by the train inside the *tc* and the point where the train signals the *SM* to move. The result produced by the tool is shown in Figure 4. The

```
Number of iterations required for
reachability: 10
     request <= train_dist &
     4train_dist <= 4request + 675
```

Figure 4: Signaling point and max. dist. traveled

variable *request* indicates the mark where the *SM* movement is requested. The variable *train_dist* indicates the distance traveled by the train inside of the track circuit. The output

produced by the tool was the following conjunction $(request \leq train\_dist) \wedge (train\_dist \leq request + 675/4)$. This clause indicates what relationship must hold between the variable *request* and the variable *train_dist* for the unsafe region to be reached. Therefore, for a safe operation, the negation of this predicate must be observed, yielding the clause, $(request > train\_dist) \vee (train\_dist > request + 675/4)$. The first disjunct never occurs, since $train\_dist \geq request$ always holds, by construction. Hence, for a safe operation, the relationship $train\_dist > request + 675/4$ must always be observed.

4.2 Timing the *SM*

Now, the objective was to determine how much slower the *SM*s could be, without compromising the safe functioning of the overall system. The time instant for the operation of a *SM* was synthesized, using variable t_3 of the *SM* automaton as a parameter. Also, if the minimum distance, with respect to the *SM*, that must be safely observed when the train signals the *SM*, is not observed the system could enter an unsafe operation state. The HyTech tool synthesized safe instants for the values of this distance.

Four experiments synthesized four different time instants for the *SM* movement. The results appear on Table 3. The first column indicates the distance inside of the track circuit where the train requests the *SM* to move. The second column shows that the maximum distance traveled by the train, up to the point when the *SM* locks, was fixed in 200 meters, corresponding to all the extension of a track circuit. The third column indicates which values for the *SM* operation time were synthesized, while still guaranteeing the safety of the system. Again, in all these experiments, the train speed was assumed to be in the interval between 7 and 9 meters per second, as indicated in the fourth column of the table.

Given the postulated speeds of the *SM*, it is clear that the last mark, counting from the

Signal (m)	Dist. (m)	Time (s)	Speed (m/s)
0	200	160/9	[7,9]
20	200	16	[7,9]
125/4	200	15	[7,9]
40	200	128/9	[7,9]

Table 3: Parametric analysis of the *SM* time

beginning of the track circuit, where the train may signal the *SM* to move is 125/4 meters. See line 3 of Table 3. If it is required of the train that it always signals the *SM* upon entering the *tc*, then a slower *SM* could be used. Line 1, in Table 3, gives the value for this slower *SM*, namely, 17.7 seconds.

The output produced by the HyTech tool, for the case illustrated in the first line of the Table 3, is shown in Figure 5. The variable *SM_time* indicates the time needed by the *SM*

```
Number of iterations required for
reachability: 5
     9SM_time >= 160
```

Figure 5: Param. analysis for the *SM* movement.

to complete its move. The relationship obtained indicates the values that cause the region of undesirable configurations to be reached. Negating this clause, the predicate that must be observed is revealed: $9SM\_time < 160$. Therefore, the clause that characterizes the safe operation is $SM\_time < 160/9$, as shown in the line 1, in Table 3.

The relationship that must hold between the *SM* movement time and the point where the train signals the *SM* to move was also synthesized. The result produced by the tool is presented in Figure 6. The variable *request* indicates the distance, counting from the

```
Number of iterations required for
reachability: 10
     request <= 200 &
     45SM_time + 4request >= 800
```

Figure 6: *SM* activation mark and its movement time

beginning of the track circuit, where the *SM* movement is requested. The variable *SM_time* indicates the time for the *SM* to operate. The output produced by the tool indicates the relationship that must be observed in order for the region of unsafe configurations to be reached. Once the trivial condition $request \leq 200$ is discarded, the negation of the new clause reduces to $45SM\_time + 4request < 800$. As a result, the predicate that must hold is equivalent to $SM\_time < (800 - 4request)/45$.

4.3 *SM* timing and the point of signaling

The two previous sets of experiments synthesized values for *SM* timing and the farthest mark from which the train can signal the *SM* to move, both in isolation. Much richer information would result from the relationship that must hold between these two parameters, while still guaranteeing the system safety. Knowing these relationship would permit the simultaneous adjustment of these parameters.

In a first run, the mark where the train signals the *SM* to move was fixed right at the beginning of the corresponding *tc*. The Figure 7 illustrates the input to the HyTech tool for this kind of parametric analysis. The region of initial configurations, *init_reg*, indicates that:

```
var
 final_reg, init_reg, reached: region;
init_reg:=
     loc[Ctrl_Train_1] = Get &
     loc[SM_3] = Normal &
     t3=0 & aux1=request;
final_reg :=
      loc[Ctrl_Train_1] = Get &
      aux1=train_dist &
      (loc[SM_3] = Normal |
      loc[SM_3] = Opening |
      loc[SM_3] = Closing);
reached:==
 reach forward from init_reg endreach;
print omit all locations
      hide non_parameters in
           reached & final_reg
      endhide;
```

Figure 7: Multiple parametric analysis

(i) train_1 is in the imminence of occupying a *tc*; (ii) the SM_3 is in the normal position; (iii) the movement time for SM_3 starts zeroed; and (iv) the parametric analysis of the distance, measured inside tc_3, of the mark where the trains signals the *SM*, will be carried out using the predicate $aux_1 = request$. The final region, *final_reg*, describes the unsafe situations, which are: (i) the train is occupying tc_3; (ii) SM_3 is not in the *reverse* position. In other words, it is either in the *normal* position, or it is going from *normal* to *reverse*, or vice-versa; and (iii) the distance of the train, measured from the initial mark of tc_3, was parameterized using the condition $aux_1 = train\_dist$. The HyTech tool determines, automatically, the region which describes all the reachable configurations from the region *init_reg*. In this example, the *Post* operator was used to compute reachability, which implies a forward analysis. After obtaining the reachable region, the tool computes the intersection of this reachable region and the region described by *final_reg*. As the region *final_reg* describes the unsafe configurations for system operation, it is necessary to negate the predicate obtained in order to reveal the relationship which must hold between the parameters in order for the system to operate safely. The result produced by the tool appears in Figure 8. The variable *train_dist* indicates the distance traveled by the train inside the *tc*. The variable *SM_time* indicates the *SM* operation time. The clause constructed by the tool is negated and the impossible condition $train\_dist < 0$ is discarded. What remains is the condition that must be observed for the system to operate safely: $SM\_time < (4/45) \times train\_dist$.

In a second run, the HyTech tool was programmed to obtain the relationship among the distance traveled by the train, the *SM* movement time, and the mark where the train

```
Number of iterations required for
reachability: 10
     train_dist >= 0 &
     4train_dist <= 45SM_time
```

Figure 8: The *SM* timing and the distance traveled

signals the *SM* to position itself, counting from the beginning of the *tc*. The result appears in Figure 9. The variable *request* indicates the distance of the mark point where the

```
Number of iterations required for
reachability: 10
     request <= train_dist &
     4train_dist <= 45SM_time + 4request
```

Figure 9: *SM* timing and signaling, and dist. traveled

SM movement is requested. The variable *SM_time* indicates the *SM* operation time, and the variable *train_dist* indicates the distance traveled by the train into the *tc*. The output produced by the tool is the relationship that must hold for the system to reach the region of unsafe configurations. Discarding of the infeasible clause $train\_dist < request$, the negation of the resulting condition gives $SM\_time < (4/45) \times (train\_dist - request)$. This condition ensures a safe operation for the segment of the subway mesh.

5 Conclusions

The daily operation of a subway mesh is a critical, real-time and reactive process. Because of its potential to cause irreparable damages, it demands a rigorous validation procedure. This work is a step in this direction.

Hybrid automata were used as mathematical tool in order to build formal models of the real system behavior. This formalism is well suited to model systems comprised of a number of individual agents that manifest a continuous, dynamic behavior, regulated by the asynchronous occurrence of discrete events. A subway mesh has all these characteristics. The continuous components are represented by the physical components of the mesh, and the discrete asynchronous events arise from the exchange of signals during communication sessions.

The formal models developed in this work were used to verify some aspects of the operational behavior of a real subway mesh segment, representing the maneuvering yard.

Operational parameters that dictate the trains relative positioning and movement, as well as some timing parameters of the switching machines were verified and synthesized. The HyTech (semi-)automatic tool was used as computational support, and it proved capable of verifying and synthesizing several aspects of the linear models that were developed. Although hindered, at times, from insufficient computational resources, all the verification and synthesis experiments were successfully programmed and run on a typical 350 MHz desktop PC, with 320MB of memory.

References

[1] R. Alur, T. A. Henzinger, and P.-H. Ho. Automatic symbolic verification of embedded systems. In *Proceedings of the 14th Annual IEEE Real-Time Systems Symposium*, pages 2–11. IEEE Computer Society Press, 1993.

[2] J. Bengtsson, K. G. Larsen, F. Larsson, P. Pettersson, W. Yi, and C. Weise. New generation UPPAAL. Technical report, BRICS, Dept. of Computer Science, Aalborg University, Denmark and Department of Computer Systems, Uppsala University, Sweden. This techincal report can be found at the URL address http://www.docs.uu.se/docs/rtmv/uppaal.

[3] J. Bengtsson, K. G. Larsen, F. Larsson, P. Pettersson, and W. Yi. UPPAAL — a tool suite for automatic verification of real–time systems. Technical Report RS-96-58, BRICS, Aalborg University, DENMARK and Department of Computer Systems, Uppsala University, Sweden, Dec. 1996.

[4] J. Bengtsson, K. G. Larsen, F. Larsson, P. Pettersson, and W. Yi. UPPAAL in 1995. *Lecture Notes in Computer Science*, 1055:111–114, 1996.

[5] A. L. Bonifácio, A. V. Moura, J. B. Camargo Jr., and J. R. Almeida Junior. Análise e Verificação de Segmentos de Via de uma Malha Metroviária. In *II Workshop of Formal Methods*, Florianópolis, Brazil, October 1999. (In Portuguese).

[6] A. L. Bonifácio, A. V. Moura, J. B. Camargo Jr., and J. R. Almeida Junior. Análise, Verificação e Síntese de Segmentos de Via de uma Malha Metroviária. Technical Report 18, Computing Institute, University of Campinas, Campinas, Brazil, August 1999. (In Portuguese).

[7] E. W. Dijkstra. Guarded commands, nondeterminacy and formal derivation of programs. *CACM*, 18(8):453–457, August 1975.

[8] J. H. Z. d. Freitas, A. Accurso, and I. L. Mathias. A evolução tecnológica da cmsp e o estado da arte de sistemas de sinalização baseado em comunicação. In *Revista Engenharia*, volume 529, pages 116–124, 1998. (In Portuguese).

[9] T. A. Henzinger and P.-H. Ho. HyTech: The cornell hybrid technology tool. *Workshop on Hybrid Systems and Autonomous Control*, Oct. 1994.

[10] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. HYTECH: the next generation. In *Proceedings of the 16th IEEE Real-Time Systems Symposium*, pages 56– 65, Pisa, Italy, 5-7, Dec. 1995. IEEE Computer Society Press.

[11] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. A user guide to HyTech. In E. Brinksma, W. Cleaveland, K. Larsen, T. Margaria, and B. Steffen, editors, *TACAS 95: Proceedings of the First Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, volume 1019 of *Lecture Notes in Computer Science*, pages 41–71. Springer-Verlag, 1995.

[12] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. HYTECH: A model checker for hybrid systems. In O. Grumberg, editor, *CAV'97: Proceedings of the Ninth International Conference on Computer-Aided Verification*, volume 1254 of *Lecture Notes in Computer Science*, pages 460–463. Springer-Verlag, 1997.

[13] T. A. Henzinger and H. Wong-Toi. Using HyTech to synthesize control parameters for a steam boiler. In J.-R. Abrial, E. Börger, and H. Langmaack, editors, *Formal Methods for Industrial Applications: Specifying and Programming the Steam Boiler Control*, volume 1165 of *Lecture Notes in Control and Information Science*, pages 265–282. Springer-Verlag, 1996.

[14] P.-H. Ho. *Automatic Analysis of Hybrid Systems*. PhD thesis, Cornell University, Aug. 1995.

[15] P.-H. Ho and H. Wong-Toi. Automated analysis of an audio control protocol. In P. Wolper, editor, *Proceedings of the 7th International Conference On Computer Aided Verification*, volume 939 of *Lecture Notes in Computer Science*, pages 381–394, Liege, Belgium, July 1995. Springer-Verlag.

[16] K. G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a NUTSHELL. *International Journal on Software Tools for Technology Transfer*, 1(1):134–152, Dec. 1997.

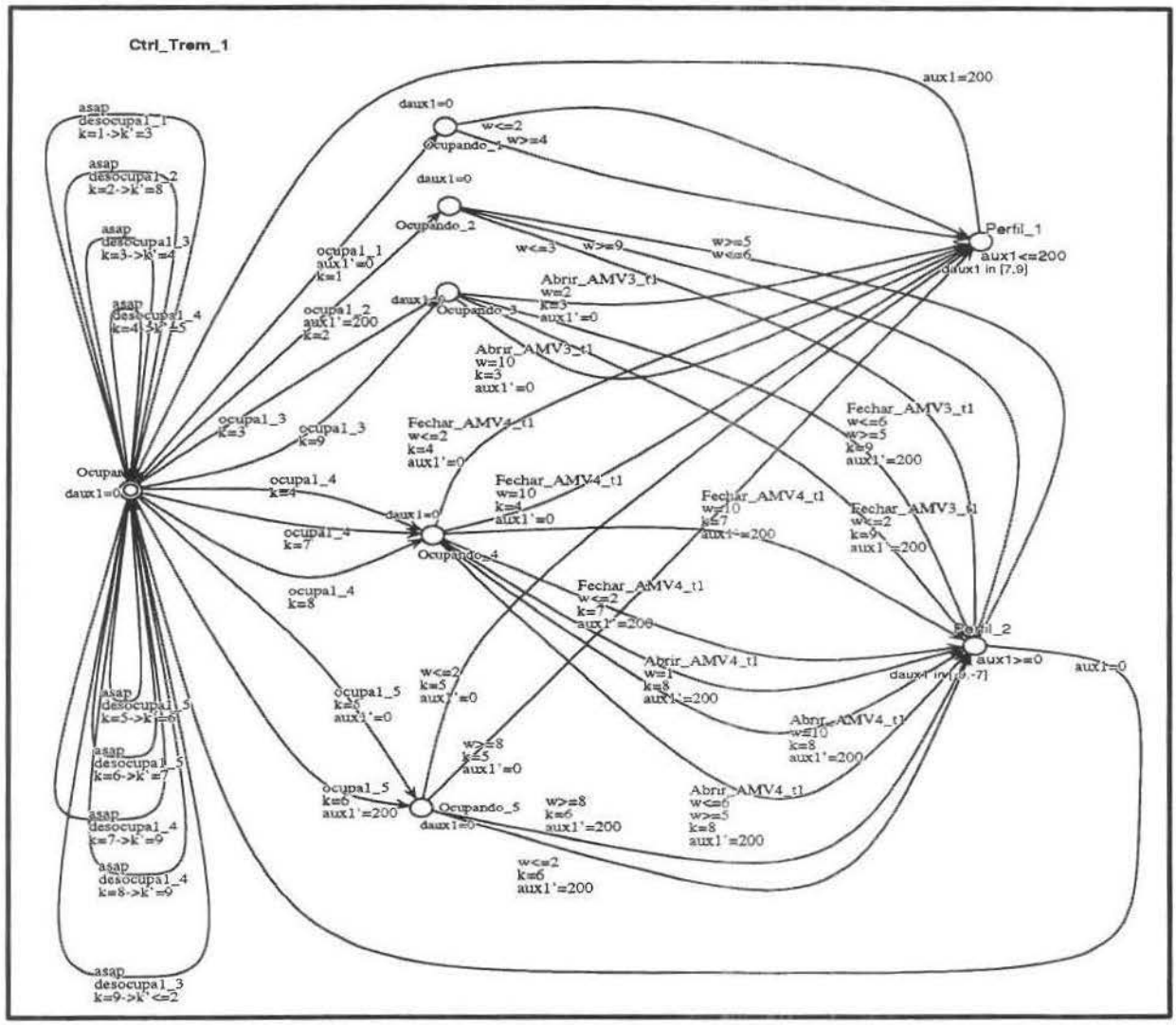


Figure 10: Train automaton

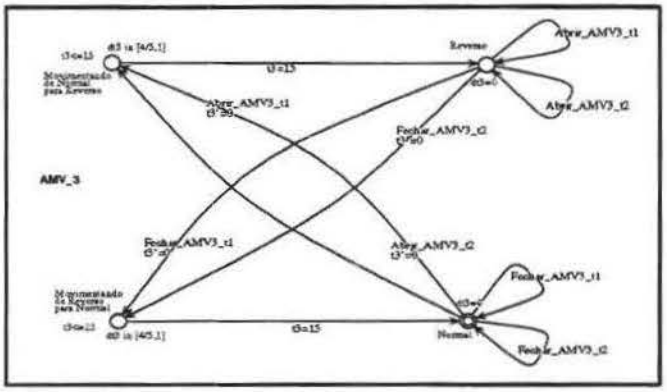


Figure 11: *SM* automaton

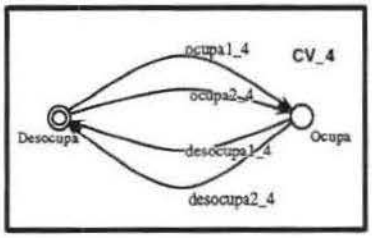


Figure 12: Track automaton

Epílogo

Neste capítulo foi apresentada, em detalhes, a verificação de propriedades para a operação segura do segmento de malha metroviária, compreendido por um pátio de manobras.

O foco esteve situado na síntese de parâmetros críticos para a operação do sistema modelado. O estudo demonstrou, novamente, a segurança dos trens que trafegam na malha metroviária, além de permitir o ajuste de certos parâmetros que influenciam diversas propriedades do sistema, as quais já haviam sido verificadas no artigo apresentado no capítulo anterior.

Os resultados alcançados compõem um primeiro passo na direção de se obter um modelo formal que descreva a operação de segmentos maiores da malha metroviária, e que considerem o movimento de mais composições simultâneas. Além disso, seria desejável acomodar também uma maior complexidade nos requisitos de segurança que são tratados. Modelos para tratar estas situações poderiam ser construídos nos mesmos moldes daqueles apresentados neste estudo. Para tal, porém, seria necessário dispor de equipamento computacional de grande porte. De outro lado, seria necessário, também, estender a funcionalidade e a capacidade da ferramenta HyTech, alterando e aprimorando o código fonte atualmente disponível. Pelo seu porte, esta tarefa está fora do escopo do presente trabalho.

Também seria de grande valia poder contar com mecanismos formais, integrados ao formalismo de autômatos híbridos, e que tratasse o problema das falhas mecânicas que podem ocorrer durante a operação do sistema, prejudicando a segurança da malha. Hoje, o formalismo de autômatos híbridos e, portanto, a própria ferramenta HyTech, não tratam este tópico. Tampouco há resultados teóricos nesta direção e que embasem a extensão do suporte oferecido atualmente pela ferramenta. Conseguir tais resultados seria uma tarefa ainda mais recompensadora, e também bem mais desafiadora.

O próximo capítulo apresenta uma abordagem, também de síntese, porém aplicado a um sistema de gerenciamento de tráfego aéreo.

Capítulo 6

Verificação e Síntese Formal para um Sistema de Tráfego Aéreo

Prólogo

O controle de aviões, cruzando um mesmo espaço aéreo, é um processo extremamente crítico. Em geral os aviões voam à altas velocidades, tornando as condições de operação delicadas e de alto risco. Nos capítulos anteriores foram enfocados estudos a respeito do funcionamento e operação de uma malha metroviária, outro cenário crítico. O estudo elaborado neste capítulo enfoca um sistema de gerenciamento de tráfego aéreo.

O capítulo é composto pelo artigo "Formal Verification and Synthesis for an Air Traffic Management System", submetido à *3rd International Conference on Formal Methods in Computer Aided Design* (FMCAD'00), a ser realizada de 1 a 3 de novembro de 2000, em Austin, Texas. O artigo tem como objetivo principal a verificação, validação e síntese de parâmetros importantes para a operação de um sistema de controle de tráfego aéreo.

O estudo aborda uma situação específica, onde dois aviões viajam num mesmo espaço aéreo. Os aviões trafegam em sentidos opostos e podem realizar manobras de mudanças de rota ao se aproximarem. Para evitar a colisão dos aviões, diferentes estudos são realizados pesquisando a segurança do sistema como um todo. Devido a complexidade dos protocolos de aproximação e do sistema de controle desenvolvido para evitar colisões, a ferramenta computacional HyTech impôs limitações ao cenário em estudo, permitindo tratar apenas o caso envolvendo manobras entre dois aviões.

Formal Verification and Synthesis for an Air Traffic Management System

Adilson Luiz Bonifácio[1], Arnaldo Vieira Moura[1],
João Batista Camargo Jr.[2], and Jorge Rady de Almeida Jr.[2]

[1]Computing Institute, Univ. of Campinas
P.O. 6176, Campinas, Brazil, 13081-970
{albonifa,arnaldo}@dcc.unicamp.br
[2]Politechnic School, Univ. of São Paulo
Av. Prof. L. Gualberto, Trav. 3, N. 158,
São Paulo, Brazil, 05508-900
{jbcjunio,jrady}@pcs.usp.br

Abstract The aim of this work is to apply formal specification techniques to model real-time distributed systems arising from real-world applications. The target system is an Air Traffic Management System (ATM), using the Traffic alert and Collision Avoidance System (TCAS) protocol. The formal models developed here are based on the notion of hybrid automata. Semi-automatic tools are used in the verification of the models, and some important system parameters are synthesized using a parametric analysis. All results were obtained on a 350MHz desktop PC, with 320MB of main memory.

1 Introduction

The use of formal specifications and verification techniques in real-world system development processes has become more and more important, especially when such processes focus on distributed systems that control critical applications, where an operational fault can cause irreparable damages. Among a wide variety of distributed systems, the ones involving reactive and real-time responses are the most difficult to model accurately. In addition, certain kind of critical applications, known as *hybrid systems*, present a dual nature, in

the sense that the system behavior is described by continuous dynamic profiles, regulated by the intervention of discrete events [5, 4, 3]. This introduces further complications and subtleties in the modeling and analysis of such systems.

Recently, the mathematical theory of hybrid automata has been gaining momentum as a good alternative to specify and verify hybrid systems [1, 10, 12, 13]. Essentially, hybrid automata are finite automata whose states include a dynamic specification of the system evolution, and whose state transitions model an abrupt change in the dynamic behavior of the system. Each component of the distributed system is modeled by an appropriate automaton. The overall system behavior is, then, captured by the corresponding product automaton [12, 13]. The communication between the system components is regulated by the exchange of messages between the independent automata. Messages are exchanged during state transitions taking place in the individual automata. Synchronism between the individual automata is also attained by the use of a shared memory, to which all the component automata have access [12, 13]. The notion of hybrid automata is adequate to model systems composed of several distributed communicating components, each one with a simple dynamic behavior. In contrast, classic control theories deal with systems with a few distributed components whose dynamic features are more complex.

In this work, hybrid automata are used to model the complexities of a real-world distributed hybrid system, consisting of an Air Traffic Management System (ATM). The ATM system coordinates many different autonomous aircraft which compete for routes in a section of airspace. The Traffic alert and Collision Avoidance System (TCAS) protocol is used to help manage the traffic in the vicinity of a cruising aircraft. The TCAS is an on-board conflict detection and resolution algorithm. Its task is to monitor the traffic around the aircraft, detect possible threats and advise on how to resolve these conflicts, for vertical separations.

The models explored in this work consider two aircraft flying in the same airspace, in opposite directions. This scenario captures only a fragment of the complexities of a real ATM system. Even so, the results obtained contribute a positive step towards the validation and the verification of the functionalities of the actual system. In all cases studied, a safe system operation is the main property to be verified. At the moment of this writing, the authors are unaware of other direct applications of hybrid automata techniques either to verify safety properties for ATM systems [14], or to synthesize values for operational parameters of such systems.

The operation of the ATM system requires an intense exchange of messages and a complex activity of coordination between all participating agents. Due to the complexity of the resulting model, semi-automatic computational tools are used to effectively bear on the formal specifications. Most of the time, models for realistic systems result in an automaton of such complexity that the cooperation of these automatic tools becomes essential. Here,

the HyTech tool [9, 11, 10] was used to analyze the models.

This work is organized as follows. Section 2 presents the notion of hybrid automata, illustrated by some simple examples. Section 3 describes the air traffic management system. The system operation is explained, together with some of its variants and special procedures. Section 4 discusses the models that were developed to verify and to validate the system operation. The synthesis of values for certain important parameters is also described here. The last section concludes with some final observations.

2 Hybrid Automata

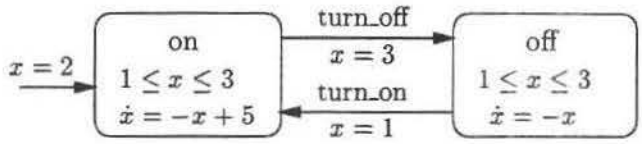
A *hybrid automaton* is a formal system $A = (X, V, \mathit{flow}, \mathit{init}, \mathit{inv}, E, \mathit{jump}, \Sigma, \mathit{syn})$, where:

1. $X = \{x_1, x_2, \ldots, x_n\}$ is a finite set of *variables*. The number n is the *dimension* of A.
2. V is a finite set of *operation modes*, or just *modes*.
3. For every mode $v \in V$, *flow(v)*, the *continuous activity condition* of mode v, is a predicate over the set of variables $X \cup \dot{X}$. Here, $\dot{X} = \{\dot{x}_1, \ldots, \dot{x}_n\}$.[3]
4. For every mode $v \in V$, *inv(v)*, the *invariant condition* for mode v, is a predicate over X. When in mode v, the invariant condition *inv(v)* must always be satisfied.
5. The *initial conditions* are given by the component *init*. For every mode $v \in V$, *init(v)* is a predicate over X. The automaton can start in mode v only if the initial condition *init(v)* is satisfied.
6. The multi-set E is formed by pairs (v, v'), where $v, v' \in V$ are operation modes. The multi-set describes the *transitions* of the automaton.
7. The component *jump* describes the *phase change conditions*. For every transition $e \in E$, *jump(e)* is a predicate over the set $X \cup X'$. Here, $X' = \{x'_1, \ldots, x'_n\}$. The primed variable x' is used to indicate the (new) value of the corresponding variable x after a transition.
8. Σ is a finite set of *event labels*, or just *events*. The partial function *syn* maps transitions in E into events of Σ.

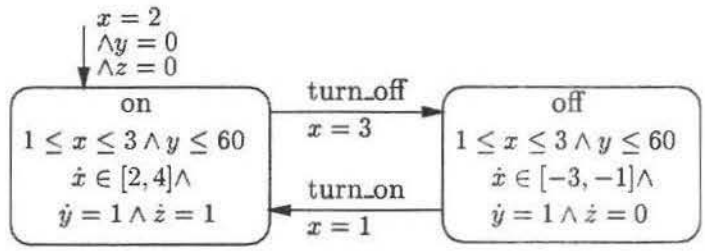
A simple example [12] of a hybrid automaton is shown in Figure 1(a). This model captures the simple dynamics of a gas heater. The model uses a real variable x, which evolves deterministically in time, to model the temperature of a container. The heater

[3] For any variable x, the first derivative of x with respect to time is indicated by $\dot{x}$.

is turned off when the temperature reaches 3 degrees, and it is turned back on when the temperature falls to 1 degree. The evolution of x is computed from the initial conditions, from differential equations present in the modes, and from the transitions between modes. In the formal set up for this example $n = 1$, $X = \{x\}$, and $V = \{on, off\}$. The continuous



(a) Simple heater



(b) Heater and safety requirement

Figure 1: Hybrid automata: an example

activity condition for mode *on* is given by the predicate $\dot{x} = -x + 5$, which describes an exponential rise of the temperature x. For mode *off* the continuous activity condition is $\dot{x} = -x$, describing an exponential fall in the temperature. For both operation modes, the invariant condition is $1 \leq x \leq 3$. The initial condition for mode *on* is $x = 2$, and for mode *off* it is `false`. The latter condition is not depicted in the figure. There are two transitions in Figure 1(a), namely (*on*, *off*) and (*off*, *on*). The transition (*on*, *off*) has a phase change condition given by the predicate $x = 3 \wedge x' = x$, and the transition (*off*, *on*) has the phase change condition given by the predicate $x = 1 \wedge x' = x$. The trivial condition $x' = x$ is not shown in the figure. In this model, the value of the real variables do not change when a transition is taken. It is common to use "guards", in the sense of [7], to represent the phase change conditions in the graphical representation of hybrid automata. A guard such as $(x_1 = x_2) \rightarrow (x_1 := 2x_2)$ gives the precondition $x_1 = x_2$ and the postcondition $x_1' = 2x_2$, that are to be enforced during a phase change. These conditions are equivalent to the complete predicate $x_1 = x_2 \wedge x_1' = 2x_2 \wedge x_2' = x_2$. The set of discrete events for this example is $\Sigma = \{turn\_on, turn\_off\}$. The function *syn* associates the event *turn_on* to

the transition (*off*, *on*), and associates the event *turn_off* to the transition (*on*, *off*). The presence of these events will allow for the synchronization between distributed automata, as will be described in the sequel.

A number of other related works, dealing with models based on hybrid automata, can be found in [9, 12, 13].

The model of a complex system comprises several individual automata that operate concurrently. The synchronism of the global system is obtained by:

1. Requiring simultaneity of phase transitions when they are labeled by the same discrete event, and

2. Using shared variables.

The synchronism is automatically incorporated into the product automaton, to be described next. Let A_1 and A_2 are hybrid automata, where $X_1 \cap X_2 = \emptyset$. The *product automaton* of A_1 and A_2 is a system $A = (X_1 \cup X_2, V_1 \times V_2, \mathit{flow}, \mathit{init}, \mathit{inv}, E, \mathit{jump}, \Sigma_1 \cup \Sigma_2, \mathit{syn})$, where the *flow*, *init* and *inv* components are simple conjunctions:

$$\begin{aligned} \mathit{flow}((v_1, v_2)) &= \mathit{flow}(v_1) \wedge \mathit{flow}(v_2); \\ \mathit{inv}((v_1, v_2)) &= \mathit{inv}(v_1) \wedge \mathit{inv}(v_2); \\ \mathit{init}((v_1, v_2)) &= \mathit{init}(v_1) \wedge \mathit{init}(v_2). \end{aligned}$$

For the phase transitions, $e = ((v_1, v_1'), (v_2, v_2')) \in E$ if and only if one of the following conditions hold:

1. $v_1 = v_1'$, $e_2 = (v_2, v_2') \in E_2$ and $\mathit{syn}_2(e_2) \notin \Sigma_1$; $\mathit{jump}(e) = \mathit{jump}_2(e_2)$ and $\mathit{syn}(e) = \mathit{syn}_2(e_2)$.

2. $v_2 = v_2'$, $e_1 = (v_1, v_1') \in E_1$ and $\mathit{syn}_1(e_1) \notin \Sigma_2$; $\mathit{jump}(e) = \mathit{jump}_1(e_1)$ and $\mathit{syn}(e) = \mathit{syn}_1(e_1)$.

3. $e_1 = (v_1, v_1') \in E_1$, $e_2 = (v_2, v_2') \in E_2$, $\mathit{syn}_1(e_1) = \mathit{syn}_2(e_2)$. In this case, $\mathit{jump}(e) = \mathit{jump}_1(e_1) \wedge \mathit{jump}_2(e_2)$ and $\mathit{syn}(e) = \mathit{syn}_1(e_1)$. The synchronism is imposed by forcing the transitions labeled by the same event to occur simultaneously.

The product of several automata is obtained by accumulating the result of computing the product of each individual automaton, in turn.

An *atomic linear predicate* is an inequality between rational constants and linear combinations of variables with rational coefficients, such as $2x + 4y - 7z/2 \leq -10$. A *convex linear predicate* is a finite conjunction of atomic linear predicates, and a *linear predicate* is a finite disjunction of convex linear predicates.

A *convex region* of a hybrid automaton A of dimension n is a convex polyhedron in $\mathbb{R}^n$. A *region* is a finite union of convex regions. Given a predicate φ, the region determined by φ is denoted by $[\varphi]$, and is called the φ-region. A *configuration* of a hybrid automaton A is a pair $q = (v, a)$ consisting of an operation mode $v \in V$ and a vector $a = (a_1, \ldots, a_n)$ in $\mathbb{R}^n$. The configuration (v, a) is *admissible* if $a \in [inv(v)]$. The configuration (v, a) is *initial* if $a \in [init(v)]$. In the example depicted on Figure 1(a), the configuration $(on, 0.5)$ is not admissible and that the configuration $(on, 2)$ is initial.

Let $q = (v, a)$ and $q' = (v', a')$ be two configurations of A. The pair (q, q') is a *phase change* of A if $e = (v, v')$ is a transition in E and $(a, a') \in [jump(e)]$. The pair (q, q') is a *continuous activity* of A if $v = v'$, if there is a nonnegative real $\delta \in \mathbb{R}$ (the duration of the continuous activity) and if there is also a differentiable function $\rho : [0, \delta] \to \mathbb{R}^n$ (the curve of the continuous activity), such that the following requirements are satisfied: (i) Endpoints: $\rho(0) = a$ and $\rho(\delta) = a'$; (ii) Admissibility condition: for all time instants $t \in [0, \delta]$, the configuration $(v, \rho(t))$ is admissible; (iii) Invariant condition: $\rho(t) \in [inv(v)]$, for all time instants $t \in [0, \delta]$; (iv) Continuous activity condition: if $\dot{\rho} : [0, \delta] \to \mathbb{R}^n$ is the first derivative of ρ, then, for all time instants $t \in [0, \delta]$, $(\rho(t), \dot{\rho}(t)) \in [flow(v)]$.

A *trajectory* of A is a finite sequence $q_0, q_1, \ldots, q_k$, of admissible configurations, where each pair (q_j, q_{j+1}) of consecutive configurations is either a phase change or a continuous activity of A. Such a trajectory is said to *start* at q_0. A configuration q' of A is *reachable from* a configuration q if q' is the last configuration of some trajectory of A starting at q. An *initial trajectory* of A is any of its trajectory that starts at an initial configuration. A configuration q' of A is simply *reachable* if it is reachable from an initial configuration of A. In the example illustrated in Figure 1(a), all admissible configurations are reachable.

Given a region φ, $Post(\varphi)$ is the region formed by all those configurations q' for which there exists a configuration $q \in [\varphi]$ such that q' is reachable from q through a continuous activity or of a phase change of A. Starting with $\varphi_0 = \varphi$, the iteration of this process will compute the regions $\varphi_{k+1} = Post(\varphi_k)$, for $k = 0, 1, \ldots$. If a region φ_k satisfies $\varphi_k = \varphi_{k+1}$, then the process converges and region φ_k contains all the configurations of A that are reachable via some trajectory that starts from a configuration in φ_0. When the process converges, region φ_k is denoted by $Post^*(\varphi_0)$. A backward process can also be defined in a similar way, giving rise to a *Pre* operator.

The following theorem can be shown to hold [1]: if A is a linear hybrid automaton, and if $[\varphi]$ is any region of A, then the calculation of $Post^*(\varphi)$ converges. This theorem supports the construction of (semi) automatic computational tools for the analysis of linear hybrid automata.

A *safety requirement* is a set of predicates imposed on the system configurations. Usually, a safety requirement is described by the set of values that the system variables can attain. A configuration is *safe* if it satisfies all the safety requirements associated with the

model; otherwise the configuration is *unsafe*. A model is *safe* if all its reachable configurations are safe; otherwise the model is *unsafe*. Given a safety requirement as a predicate φ over the configurations of A, the region $reach = Post^*([init])$ is computed, and the intersection $reach \cap [\neg\varphi]$ is obtained. If the resulting region is empty, the safety requirements are satisfied and the system is safe; otherwise they are violated and the system is unsafe.

System parameters are symbolic constants which assume fixed values. A parametric analysis determines value intervals for certain system parameters in such a way as to guarantee that no safety requirement is violated. This process, therefore, synthesizes maximum and minimum values for these parameters. In a hybrid automaton, a parameter α can be represented by a variable whose value never changes. This condition can be attained by imposing the condition $\dot{\alpha} = 0$ in all operation modes and, further, by imposing the condition $\alpha' = \alpha$ over all state transitions of the automaton. A value $a \in \mathbb{R}$ is *safe for a parameter* α if no unsafe configuration is reachable when the initial condition $\alpha = a$ is adjoined to the all operating modes of the automaton.

Returning to the heater example, depicted in Figure 1(a), a safety requirement for the heater could be: "in the first hour of operation, the heater should not be on by more than 2/3 of the time". Figure 1(b) depicts a relaxed version of that automaton, where the new clock variable y measures the total elapsed time and the new stopwatch variable z measures the time the heater stays in operation. It can be demonstrated that the relaxed version is safe with respect to these requirements [12]. It follows that the original model is also safe with respect to the same requirements.

In general, the modeling of realistic systems results in a product automaton of such complexity that the verification phase can only be successfully done with the aid of (semi) automatic computational tools. The software HyTech [9, 10, 11, 12] was developed as a tool to aid in the analysis and verification of linear hybrid automaton models. The model is described to the tool via an input file that comprises two parts. The first part describes the individual hybrid automata. The product automaton is computed automatically by the tool. The second part is a sequence of analysis commands.

The HyTech tool can automatically verify that the relaxed heater model illustrated in Figure 1(b) satisfies all the posed safety requirements [12].

Today, HyTech is one of the only tools that supports semi-automatic analysis of linear hybrid automata models. It has been used in a variety of diverse situations and further examples can be found in [1, 9, 12, 11].

UPPAAL [2] is another example of an automatic verification tool. It is based on timed-automata, a more restricted notion than that of linear hybrid automata. The former notion can only work with clocks, and does not support the use of more complex differential equations to describe dynamic system profiles.

3 Description of an Air Traffic Management System

Air Traffic Management Systems (ATM) are very complex and critical systems, that operate under tight conditions and are composed of several distributed components. This makes it hard for such systems to be formally verified. On the other hand, the need for a formal, rigorous verification of all the system safety requirements is unquestionable, since any safety fault can result in intolerably high damages, both in terms of properties and lives. Several faults are known to have occurred in aviation history, causing terrible accidents, due the unsafe operation of these systems [16].

In a typical flight, after the aircraft enter en route, the Air Traffic Control (ATC) monitors the flight by radio [8]. After an ATC acknowledges radar contact with the signaling plane, it starts to transmit values for heading and altitude, based on the aircraft present coordinates. Depending on the flight plan, one aircraft can switch many times from one en route controller to another, during the course of its flight. En route controllers are assigned to specific geographic areas, and they work to maintain the safe separation of passing aircraft that cross their sector of airspace. While all airline aircraft are controlled every step of the way, the same level of positive control does not always apply to all other aircraft. Some aircraft can, and often do, fly in uncontrolled airspace, outside the ATC range. In general, these uncontrolled air spaces are areas below the cruise lanes used by airline aircraft.

General aviation aircraft are allowed to fly under visual flight rules, or VFR, when weather and visibility are good. In these conditions, they do not have to file a flight plan, and they do not have to be in touch with air traffic control, unless they choose to operate in or out of an airport that has a control tower. Under VFR rules, pilots are responsible for maintaining adequate separation from other aircraft, and that is why these rules are sometimes called the "see and be seen" rules.

Instrument flight rules, or IFR, on the other hand, are the rules under which general aviation aircraft must fly in bad weather and low visibility. In this case, pilots must be in contact with the ATC and must file a flight plan. They also must be "instrument certified", meaning that they are proficient at navigating and flying their aircraft using cockpit instruments only, without the benefit of good visibility. Airline flights always operate under instrument flight rules, regardless of weather.

Recent technological advances made it possible to implement sophisticated functions, such as navigation and aircraft separation, in the system components responsible for the control and surveillance of a sector of airspace [17]. This more advanced technology permits a reduction in the number of collision, while efficiently maintaining the throughput of the airspace with adequate levels of security and reliability. On the other hand, the use of more advanced technologies tightens further the system parameters and accrues the verification

and safety problems, opening up the possibilities for the introduction of subtle errors. In these cases, a formal verification of the system safety is even more desirable.

Over the years, air traffic has continued to increase. The developments of modern air traffic control systems have made it possible to cope with this increase, while maintaining the necessary levels of flight safety. However, the risk of airborne collision remains. That is why the concept of an airborne collision avoidance system has been considered, giving rise to the initial development of the Traffic alert and Collision Avoidance System (TCAS) [16]. TCAS is a protocol used to monitor air traffic in the vicinity of flying aircraft. It provides the pilot with information about neighboring aircraft that may pose a collision threat and, also, it advises on how to resolve these conflicts. TCAS significantly improves flight safety. However, it can not entirely eliminate all collision risks. As in any predictive system, it might itself induce a risk of collision [14].

The TCAS system can enter in one of two levels of alertness [15]. In the first level, the system issues a Traffic Advisory (TA) signal to inform the pilot of a potential threat. When operating in this level of alertness, it does not provide any suggestions on how to resolve the situation. If the risk of collision increases, a Resolution Advisory (RA) signal is issued, and the system also suggests a maneuver that is likely to resolve the conflict. The TCAS system also allows for reversal commands and it may change the climb/descend advise during a conflict. This feature was added to the protocol in order to compensate for the nondeterminism in the pilot response. That is, the pilot may choose not to follow the TCAS advises thereby rendering the original maneuver unsafe. The TCAS detects this situation and changes the RA, if necessary. Obviously, this nondeterminism renders the necessity of a formal verification of the overall system safety even more necessary.

Aircraft separation standards vary according to circumstances. Above 29000 feet, when the aircraft are cruising at high speed, the standard is five miles of horizontal radar separation and 2000 feet of vertical separation. Below 29000 feet, the vertical separation is reduced to 1000 feet while the horizontal radar separation remains at five miles. When aircraft are moving at much slower speeds, as when they depart or approach an airport, the standard is three miles of horizontal radar separation and 1000 feet of vertical separation.

The TCAS protocol is designed to ensure collision avoidance between any two aircraft, with an approximation speed of up to 1200 knots and vertical rates as high as 10000 feet per minute. The controller action begins when the aircraft horizontal separation is 5 miles. The standard value for vertical climbs or descends is 10000 fpm. In emergencies it can be as high as 12000 fpm.

The TCAS system monitors the vertical plane, and only a few works, today, consider the parallel separation between the aircraft. In the future, the TCAS system might produce RAs for both the horizontal and the vertical planes [18].

Aviation texts, usually, do not treat measures in a uniform way. For example, the

horizontal speed may be given in miles per hour and the vertical rate may be measured in feet per minute. Or, the vertical distance may be presented in feet and time may be given in minutes or hours, while the horizontal distance is given in miles. See Figure 2. In this work, however, all computed measures are converted to the standard metric system, so as to maintain a uniformity throughout. Hence, the horizontal and vertical distances are given in meters and the time is measured in seconds[4].

The emphasis of this work is on the verification and synthesis of some safety requirements of an air traffic management system. The aim is to validate the system operation, and to synthesize more appropriate and tighter values for some critical system parameters, in such a way to improve the system overall performance and to decrease the occurrence of unsafe situations. One of the main aspects of the modeling is to capture the cooperation between the system parts.

Figure 2 depicts a situation with two aircraft in the same airspace. The changes of

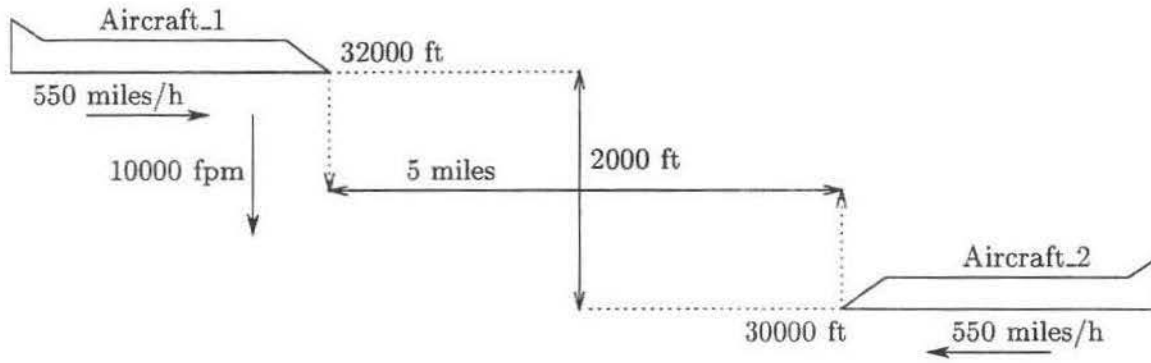


Figure 2: Two aircraft in the same airspace

attitude for the aircraft are provoked by the controller that monitors the vertical and horizontal separations between the aircraft. Based on the position between the two aircraft, the on-board TCAS controller may command the leftmost aircraft to climb or to descend. It may also increase its descent rate, if the situation is critical. The rightmost aircraft is assumed to remain in a leveled cruise. The controller may, however, reduce its horizontal speed in order to avert collisions.

In the scenario depicted in Figure 2, the leftmost aircraft is traveling at a height of 32000 feet and the rightmost one is traveling in the opposite direction, at a height of 30000 feet. The vertical distance between the aircraft is 2000 feet[5]. The aircraft are supposed to

[4] 1 foot $\simeq$ 0.3048 meters, 1 mile $\simeq$ 1829 meters and 1 knot $\simeq$ 0.51 meters per second.

[5] 32000 feet is about 9754 meters; 30000 feet is about 9144 meters; and 2000 feet is about 610 meters.

travel at 550 miles per hour, when cruising. The speed of 490 miles per hour[6] can also be applied in order to reduce the cruise speed. These rates and values are based on real data for Boeings 707 and Boeings 747[7].

The TA signal was suppressed from the models treated here and, when a conflict situation arises, a RA signal is issued directly. On the other hand, it is relatively easy to include more details of the TCAS protocols in the models, or to include more aircraft in the scenario. This would result in a more complex product automaton. The HyTech tool, however, was already operating close to its limits, running on a 350MHz desktop PC with 320MB of main memory. Certain internal limits of the tool, signaled by multiplication overflow errors, were also being reported. In order to accommodate a more complex and more complete model, it would be necessary to modify the source code of the tool and port it to a more powerful hardware platform.

4 Parameter Synthesis

The system modeled in this work comprises two aircraft flying in opposite directions, as depicted in Figure 2. The models capture several kinds of maneuvers that the aircraft can perform in the same airspace. The automaton model for the aircraft that is flying from left to right is more complex than the model for the other aircraft, which is flying in the opposite direction. This is because, at some instants, the first aircraft may decide to engage in some kind of specific maneuver, while the second aircraft is assumed to remain cruising en route. Clearly, in order to cope with such behavior, more operation modes are necessary to specify a model for the first aircraft. The full system model comprises three individual hybrid automata: one for each aircraft and another one for the system controller.

All the case studies reported here focus on the overall system safety. For each case, an analysis of the model is conducted and some parametric synthesis of critical values, such as the aircraft relative height and horizontal separation, is also performed.

4.1 The Hybrid Automata Models

The model for the leftmost aircraft is depicted in Figure 3, and the model for the rightmost aircraft is shown in Figure 4. Variable x_i indicates the horizontal distance and variable y_i indicates the vertical position of the aircraft. Variable k is used to extract information about the direction of flight, as will be explained later. The $x_1 = x_2$ horizontal mark is the position where both aircraft cross, and it is called the *critical point*. Note that the absolute position of the critical point varies, since it depends on the relative horizontal speed of both

[6] 550 miles per hour $\simeq$ 280 meters per second; and 490 miles per hour $\simeq$ 250 meters per second.

[7] http://www.air-transport.org/public/Handbook/

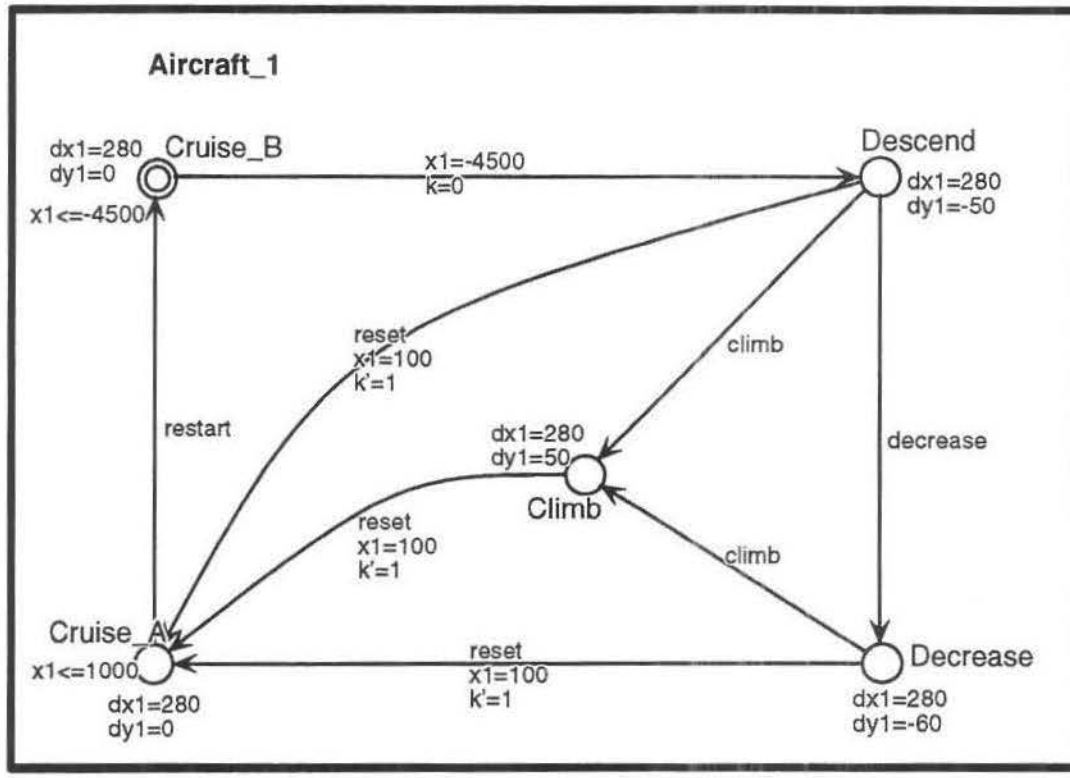


Figure 3: A model for the leftmost aircraft

aircraft. Hence, the critical point is not always at the zero horizontal mark. A negative value for the horizontal position is measured to the left of the zero horizontal mark, while positive values indicate distances to the right of this point. At the beginning, both aircraft are symmetrically positioned at 6000 meters from the zero mark. The leftmost aircraft is cruising at 9750 meters and the rightmost one is cruising at 9140 meters.

The automaton for the leftmost aircraft starts off in the "Cruise_B" mode, as shown in Figure 3. From this mode, at the mark of 4500 meters, it enters in the "Descend" mode and starts to lose altitude at a rate of 50 meters per second[8]. From this point, a first nondeterministic alternative calls for a scenario of no interaction between both aircraft. This is captured by the transition that goes directly from the "Descend" mode to the "Cruise_A" mode. This transition happens at 100 meters from the critical point (indicating that both aircraft have already passed by each other) and it uses the "reset" event to synchronize with the models for the other aircraft and the controller. While in the "Descend" mode, the leftmost aircraft faces two other nondeterministic alternatives. It might receive a com-

[8] About 10000 feet per minute.

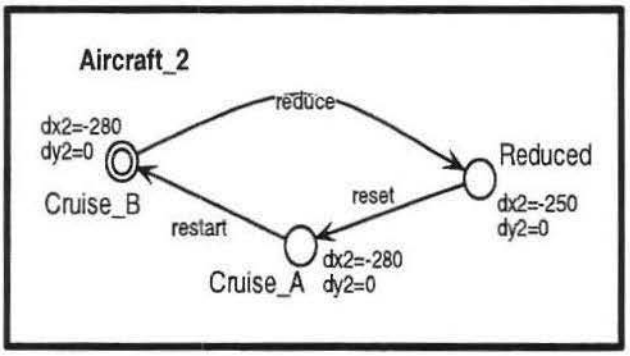


Figure 4: Aircraft_2 automaton

mand to climb, moving into the "Climb" mode, synchronizing on the "climb" event, or it might receive a command to increase its descending rate, entering the "Decrease" mode and synchronizing on the "decrease" event. At the "Decrease" mode, there are two nondeterministic choices. Either, as a result of the interaction of both TCAS, the aircraft receives a counter-order to start climbing up, moving to the "Climb" mode, or it might fly by the second aircraft while still descending. In the first case, the leftmost aircraft will fly by the second aircraft while still climbing up. In any case, from the "Climb" mode or from the "Decrease" mode, a transition to the "Cruise_A" mode is taken, at a horizontal position of 100 meters, signaling that the fly by has already happened. In both cases, a "reset" event is issued, in order to synchronize this move with the model that describes the behavior of the rightmost aircraft. Finally, at the "Cruise_A" mode, the first aircraft assumes a leveled flight and returns to the start mode no later than when it passes by the 1000 meters mark, to the right of the critical point. The horizontal speed for this aircraft is kept constant at 280 meters per second[9].

The second aircraft is modeled by a simpler automaton, composed of three modes, as shown in the Figure 4. The automaton starts off in the "Cruise_B" mode, positioned at 6000 meters to the right of the zero horizontal mark and cruising at 9140 meters. Initially, the aircraft horizontal speed is 280 meters per second, traveling from right to left. From this mode the aircraft might, upon detecting the "reduce" event, move to the "Reduced" mode, where its horizontal speed decreases to 250 meters per second. The aircraft travels at this speed until it detects the "reset" event and synchronizes with the automaton that controls the first aircraft, moving to the "Cruise_A" mode. It remains at this mode until it reaches the critical point, when both aircraft cross each other. Next, the automaton returns to the start mode, upon detecting the "restart" event.

[9] About 550 miles per hour.

The model of the controller implements (a simplified version of) the TCAS protocol, and is illustrated in Figure 5. The automaton starts off in the "Normal" mode, and it may stay

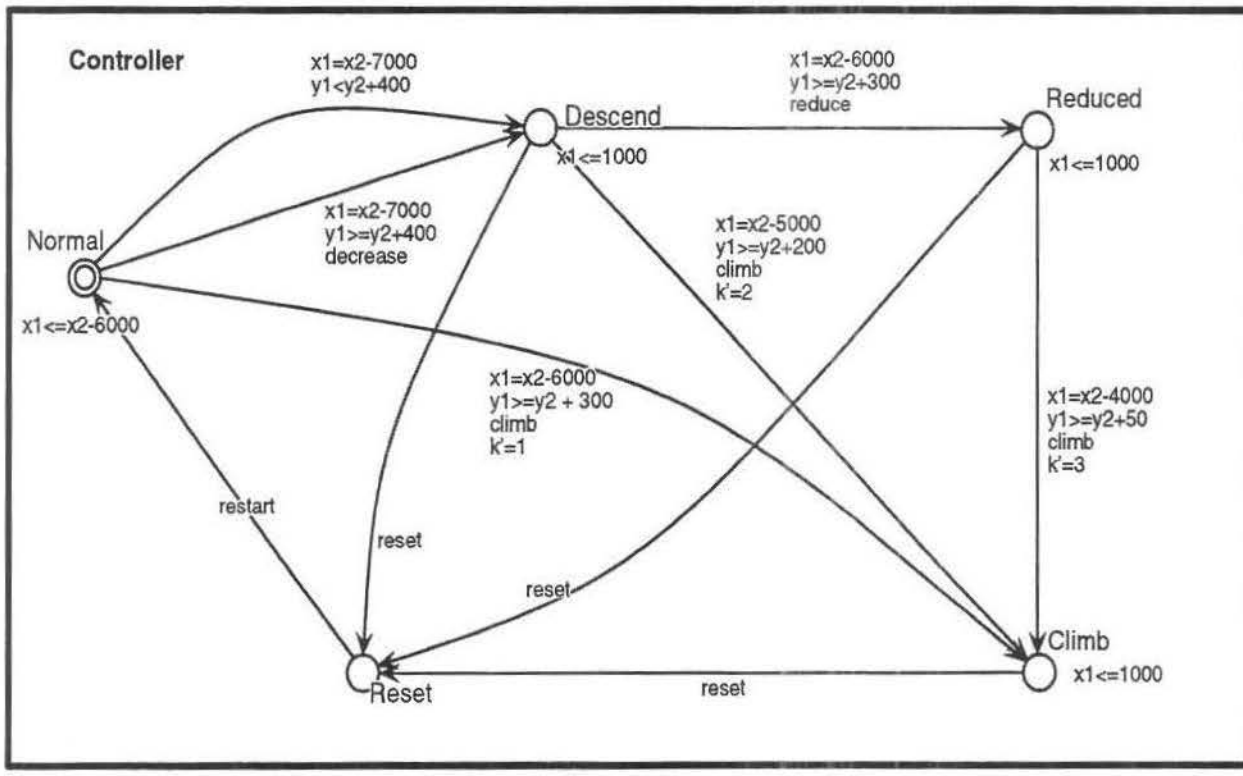


Figure 5: Controller automaton

there until the horizontal separation between the two aircraft drops to 6000 meters. But earlier, at 7000 meters of horizontal separation, a nondeterministic choice occurs. Either the controller remains in the "Normal" mode, or it switches to the "Descend" mode. In the latter case, a synchronizing "decrease" event occurs if the vertical separation between the aircraft is at least 400 meters[10]. This event forces the first aircraft to increase its rate of descent.

If the nondeterministic choice is not exercised at 7000 meters, the controller continues in the "Normal" mode, monitoring the horizontal and vertical separation between both aircraft. When the horizontal distance drops to 6000 meters, and if the vertical separation is at most 300 meters[11], a "climb" event forces the first aircraft into a 50 meters per second[12]

[10] About 1315 feet.
[11] About 985 feet.
[12] About 10000 feet per minute.

climbing trajectory, since it is not safe to continue the descent and cross the second aircraft route. As a result of the transition, the controller reaches the "Climb" mode.

If the nondeterministic choice is taken at the mark of 7000 meters, then the controller may stay in the "Descend" mode until the aircraft have crossed each other and the (originally) leftmost one has moved up to 1000 meters to the right of the zero point. In the "Descend" mode, one of three other nondeterministic choices will prevail. First, the controller may issue a "reset" event and move into the "Reset" mode, where it stays until the first aircraft has passed the zero mark by 100 meters. Or the controller might decide to act when the horizontal separation between both aircraft has reached the mark of 6000 meters, their vertical separation being at least 300 meters, or the controller might act when the horizontal separation between the aircraft has reached the mark of 5000 meters, with the vertical separation between them being at least 200 meters. In the latter case, a "climb" event is issued, forcing the first aircraft in an ascending trajectory, and moving the controller to the "Climb" mode. In the first case, a "reduce" event slows down the rightmost aircraft and puts the controller into the "Reduced" mode.

After the controller reaches the "Reduced" mode a similar scenario evolves, with the controller monitoring the vertical and horizontal separation between both aircraft. The controller may stay in this mode until the aircraft have crossed and the first one has passed the zero point by 100 meters. Or, at a horizontal separation of 4000 meters, if the vertical separation between both aircraft is at least 50 meters, a "climb" event is issued and the controller passes to the "Climb" mode.

From the moment that the "Climb" mode is entered, the controller stays there until a "reset" event happens, when the first aircraft has passed the zero point by 100 meters. See also Figure 3. At this moment, the controller jumps to the "Reset" mode. In this mode, the controller waits until the aircraft have crossed and the (originally) leftmost one has traveled a distance not exceeding 1000 meters from the zero point. At the "Reset" mode, the automaton waits for the "restart" event and moves on to the "Cruise_B" mode, restarting the cycle.

The automaton that governs the overall system behavior is obtained by calculating the product of the three individual automata.

4.2 Case Studies

The system behavior was captured and analyzed by running a number of different experiments. All case studies discussed in the sequel focus on establishing the safety of the system operation. To this end, several parametric synthesis were run. Experiments of this kind can reveal the limits of a safe system operation and can also be instrumental in obtaining tighter values for a number of system parameters.

All experiments were run using the HyTech computational support tool[13]. Figure 6 illustrates a typical input to the HyTech tool for a parametric analysis. The region of

```
var
     final_reg, init_reg, reached: region;
init_reg:=
     loc[Aircraft_1] = Cruise_B &
     loc[Aircraft_2] = Cruise_B &
     loc[Controller] = Normal &
     x1=-6000 & x2=6000 & y1=9750 & y2=9140 & k=0;
final_reg :=
     loc[Aircraft_1] = Decrease &
     loc[Aircraft_2] = Reduced &
     loc[Controller] = Reduced &
     x1=x2 &
     y1<=height;
reached:=
      reach forward from init_reg endreach;
print omit all locations
      hide non_parameters in
           reached & final_reg
      endhide;
```

Figure 6: Height parametric analysis

initial configurations, *init_reg*, indicates that:

1. *Aircraft_1* and *Aircraft_2* are in their respective start modes, "Cruise_B";
2. The *Controller* is in its start mode, "Normal";
3. The initial position for both aircraft are specified next, in meters[14];
4. The auxiliary variable k is zeroed. This variable indicates the mode of origin when the "Climb" mode is entered.

The particular parameter values used in this exercise are typical. They could easily be changed in order to reflect different values. An example of a final region, *final_reg*, capturing an unsafe region according to the TCAS protocol, is described next in Figure 6. It comprises five lines:

1. *Aircraft_1* is in the "Decrease" mode, indicating a steep descent;
2. *Aircraft_2* is in the "Reduced" mode, indicating a reduced horizontal speed;

[13] http://www-cad.eecs.berkeley.edu/ tah/HyTech/

[14] 6000 meters is about 3.3 miles; 9750 meters is about 32000 feet; and 9140 meters is about 30000 feet.

3. The *Controller* is in the "Reduced" mode, reflecting the two conditions above;
4. Both aircraft are at the same position, that is, they are passing by each other;
5. The last condition parameterizes the vertical distance of the leftmost aircraft.

The next line, in Figure 6, asks the HyTech tool to perform a forward analysis in order to compute the *reached* region. The last four lines specify a parametric print out of the region of points belonging to *reached* and *final_reg* regions. This intersection will contain all unsafe points for the system, since *final_reg* describes the region of unsafety. By negating the region calculated by the tool, it is possible to obtain the region of safe values for the *height* parameter.

All computational results were obtained by running the HyTech tool on a typical 350MHz Pentium II PC, with 320MB of main memory. Memory usage was never a problem when running the experiments. Also, for each experiment, the time consumed to complete the analysis was always quite reasonable, in the range of a few seconds, with the *Post* operator converging after a few iterations. In contrast, any attempt to increase the number of aircraft caused both the forward and backward analysis to fail. These observations indicate that the HyTech tool was operating close to its limit, and a careful construction of the models had to be undertaken, always working at these limits.

In all experiments, command line options were used in order to avoid library overflow errors, caused by multiplication of large integers. The output message "*Will try hard to avoid library arithmetic overflow errors*", produced when running the HyTech tool, indicates the use of command line options to avoid such errors. Another point to be mentioned is that all backward analyses failed due, again, to multiplication overflow.

In the rest of this section, six case studies are presented, reporting on the synthesis of values for several critical parameters. More details about each of the experiments can be found in [12].

The minimum height before the controller action

This study focuses on the minimum height reached by the first aircraft, in the worst scenario and before the controller starts to enforce the TCAS protocol. To capture this situation, the final region is specified by positioning the automaton for the first aircraft in the "Descend" mode, the automaton for the second aircraft in the "Cruise_B" mode, and the automaton for the controller in the "Normal" mode. Note that, because of the synchronizing "reset" event, it is not possible for the controller model to complete a full cycle before the aircraft models also restart at the initial mode. The output of the HyTech tool for this experiment is shown in Figure 7. The parametric analysis shows that the first aircraft can reach a

minimum height of 9482 meters[15] before the controller starts to act. Observe that this vertical distance is still above the cruising altitude of the second aircraft. This indicates that the fly by is unsafe.

```
Will try hard to avoid library arithmetic overflow errors
Number of iterations required for reachability: 7
     7height >= 66375

Time spent = 0.63 sec total
```

Figure 7: Parametric analysis before the controller action

At critical point with increased descend

This case study investigates the safety condition at the critical point when the first aircraft has taken the decision to increase its descend, while the other aircraft is still unaffected. The situation is specified by letting the automaton for the first aircraft enter the "Decrease" mode, while keeping the second automaton in the initial "Cruise_B" mode. The automaton for the controller moves to the "Descend" mode, forced by the "decrease" event, and stays there. That is, the controller does not engage in climbing maneuvers, nor commands the second aircraft to reduce its cruising speed. See Figure 3 and 5. The extra condition specified by $x_1 = x_2$ guarantees that the final region is focusing on the critical point. Again, the minimum height reached by the first aircraft was synthesized using the *height* parametric variable.

The HyTech tool synthesized the conditions that must be observed for this situation to occur. The minimum height value reached by the first aircraft is 8821 meters, or 28940 feet. From the point of view of the second aircraft, which is cruising at an altitude of 9140 meters, or 30000 feet, this value is inferior to the minimum acceptable distance of 2000 feet, as required by the protocol. However, from the point of view of the first aircraft, which is now below 29000 feet, the synthesized minimum value of 28940 feet is still within the acceptable interval of up 1000 feet for the vertical separation between both aircraft.

In the same situation, that is, with the three automata reaching the same final modes, another study was conducted. Note that the controller automaton passes from the "Normal" to the "Descend" mode when the horizontal distance between both aircraft is 7000 meters, and the vertical distance between them is at least 400 meters[16]. In this second exercise, the 400 meters separation was parameterized by the *diff_height* variable. This was

[15] About 31110 feet.
[16] About 1310 feet.

accomplished by changing the condition $y_1 \geq y_2 + 400$ into $y_1 \geq y_2 + \mathit{diff\_height}$, in the controller automaton, shown in Figure 5. The HyTech tool, then, synthesized the maximum vertical separation between both aircraft, at the moment when the first aircraft starts its increased descent, given that this change of behavior takes place exactly when the aircraft are 7000 meters apart. The result produced by the tool appears in Figure 8. It indicates

```
Will try hard to avoid library arithmetic overflow errors
Number of iterations required for reachability: 4
     7diff_height = 3020    & 7height >= 61750

Time spent = 0.15 sec total
```

Figure 8: Parameterizing vertical distances

that the maximum relative vertical distance between the aircraft, for this situation to occur, can be 431 meters[17]. The minimum height reached by the first aircraft, at the crossing point, is still 8821 meters.

It is also as easy to parameterize the horizontal separation between both aircraft, while maintaining the minimum vertical separation at 400 meters between them, both measured at the point when the first aircraft starts to increase its descent. Figure 9 presents the output of the HyTech tool for this exercise, where the parameter *diff_horiz* indicates the desired minimum horizontal distance. The synthesis revealed that the command to increase

```
Will try hard to avoid library arithmetic overflow errors
Number of iterations required for reachability: 4
     diff_horiz = 6648    & 7height >= 61794

Time spent = 0.15 sec total
```

Figure 9: Parameterizing the horizontal and vertical distances

the descent should be issued no later than the point where the horizontal separation is 6648 meters. In this case, observe that the height reached by the first aircraft is now slightly higher, reaching 8827 meters[18], which is a little unsafer.

[17] About 1414 feet.
[18] About 28960 feet.

Reducing horizontal rate and no climbing

The next experiment will allow for the second aircraft to reduce its horizontal speed. For this to occur, the final region is specified as in the previous case, except that the automata for both the second aircraft and the controller are now allowed to synchronize on the "reduce" event, and proceed to reach the "Reduced" mode.

The same three situations as in the previous subsection were analyzed here. First, the HyTech tool synthesized the minimum vertical distance for the first aircraft, without parameterizing the coordinate values where it starts its increased descent. The value obtained was 8785 meters[19], which is a slightly safer value than the one revealed in the previous section, but the difference it is not significant. Or either, the reduction in the horizontal speed for the second aircraft must be more substantial in order to have a greater influence upon the system safety conditions.

Next, the HyTech tool was run, for the same scenario, and when the 300 meters minimum vertical separation is also synthesized, using the parametric value *diff_height*. This variable measures the vertical separation between the aircraft, when the second aircraft receives the command to reduce its speed. The horizontal separation at this point is maintained at 6000 meters. The result shows that the vertical separation can be relaxed to 324 meters, a higher value.

In contrast, the HyTech tool was unable to converge, due to library overflow errors caused by multiplication operations, when trying to parameterize the minimum horizontal separation. Neither a backward nor a forward analysis could be completed, in this case.

At the critical point and climbing

Next, the situation where the first aircraft is ordered to climb is analyzed. In this initial case study, the first aircraft does not increase its descent, nor does the second aircraft reduces its cruising speed. The final region for the models is given by requiring the automaton for the first aircraft to reach the "Climb" mode directly from the "Descend" mode. The automaton for the second aircraft remains in the initial "Cruise_B" mode, and the controller automaton moves directly from the "Normal" mode to the "Climb" mode. All the evaluations are still being taken at the critical crossing point. Note that, here, the value $k = 1$ is also part of the critical region. This guarantees that the automata are following the transitions as specified. That is, the first aircraft is descending, but not at an increased rate, and the second aircraft is not reducing its horizontal speed. See also Figure 3 and 5.

The result shows that the maximum height reached at the critical point, while the first aircraft is climbing, measures 10017 meters[20]. Note that this height is greater than the

[19] About 28822 feet.

[20] About 32865 feet.

initial cruising height of 32000 feet. This shows that aborting the descent, in this case, could be premature. The next experiments provide tighter values for these parameters.

First, the vertical separation of 300 meters is parameterized, on the transition from the "Normal" mode to the "Climb" mode, in Figure 5. The parameter *diff_height* replaces the value 300 meters in the condition $y_1 \geq y_2 + 300$. The value of 342 meters is returned by the HyTech tool for the vertical separation parameter. Since the horizontal distance between both aircraft was maintained at 6000 meters, the maximum vertical distance reached by the first aircraft while climbing was still the same 10017 meters.

When parameterizing the horizontal separation between both aircraft, at the point where the climb command is issued, the results returned by the tool are as shown in Figure 10. The vertical separation is maintained at 300 meters. As can be seen, now a linear relation-

```
Will try hard to avoid library arithmetic overflow errors
Number of iterations required for reachability: 8
     28height >= 5diff_horiz + 250500    & diff_horiz <= 9000
     & diff_horiz >= 5528

Time spent = 0.21 sec total
```

Figure 10: Horizontal separation and climbing

ship holds between the horizontal separation, when the maneuver is aborted by the climb command, and the vertical distance reached by the first aircraft, at the crossing point. The condition $diff\_horiz \leq 9000$ is trivially observed, given that the first aircraft starts to descend when the horizontal separation between them is exactly 9000 meters. The linear relationship is, then, reduced to

$$(28height \geq 5diff\_horiz + 250500) \wedge (diff\_horiz \geq 5528).$$

The last clause, $diff\_horiz \geq 5528$, indicates that the minimum horizontal separation between the aircraft can be reduced to 5528 meters. In that case, the other clause says that the maximum climbing point, for the first aircraft, can reach 9933 meters[21]. This shows that, even if the first aircraft goes into a descent and then the maneuver is aborted at its minimum possible horizontal separation distance, the first aircraft can still reach a point higher than its original cruising altitude. In the worst case, the vertical separation between both aircraft, at the crossing point, is greater than 32000 feet and, as a consequence, the maneuver is deemed a safe one.

[21] About 32588 feet.

Increasing the descent, then climbing

This is the same situation as in the previous subsection, except that the first aircraft has already engaged in a steeper descent route. Here, the final region is modified only by changing the condition on the k variable. Note that the vertical separation at the point where the normal maneuver is interrupted by the climbing command is now at 200 meters, while the horizontal separation between both aircraft is set at 5000 meters, as dictated by the TCAS protocol.

The maximum height reached by the first aircraft, while climbing in this situation, is 9803 meters[22]. Observe, that this height is still greater than the initial cruising height of 32000 feet, but the difference is now smaller, when compared to the height reached in the maneuver studied in the previous experiment.

Next, the vertical separation of 200 meters was parameterized. The horizontal separation, in this case, remained at 5000 meters mark. The relaxed maximum value obtained was 217 meters, at the point where the climb command is issued.

As in the previous test cases, the horizontal separation between the aircraft, when the climb command is issued, was also synthesized. The vertical separation was maintained at 200 meters. Figure 11 illustrates the output of the HyTech tool for this case. The relation $diff\_horiz \leq 7000$ is trivially observed, since the first aircraft starts its descent already at the 7000 meters mark. Ignoring this clause, the conjunction computed by the tool reduces to the linear relationship

$$(56 height \geq 11 diff\_horiz + 494000) \wedge (diff\_horiz \geq 4840).$$

Hence, the maneuver can be aborted as late as when the horizontal separation between the aircraft reaches 4820 meters. Even in this extreme case, the height reached by the first aircraft, at the crossing point, is 9772 meters[23], still allowing for the minimum of 2000 feet required for a safe operation.

Increasing the descent, reducing the horizontal rate and aborting

In the last experiment, the first aircraft starts its descent, at a vertical rate of 50 meters per second, then it maneuvers to increase its downward vertical rate to 60 meters per second. The second aircraft, sensing the presence of the other aircraft, reduces its horizontal rate from 280 meters per second to 250 meters per second. When both aircraft are 4000 meters apart, the maneuver is aborted by the controller, and the first aircraft receives a climb command.

[22] About 32163 feet.
[23] About 32060 feet.

```
Will try hard to avoid library arithmetic overflow errors
Number of iterations required for reachability: 7
     56height >= 11diff_horiz + 494000   & diff_horiz >= 4840
     & diff_horiz <= 7000

Time spent = 0.18 sec total
```

Figure 11: Horizontal separation and vertical distance with increased descent, followed by a climb

The HyTech tool computed that the first aircraft reaches a minimum height of 9615 meters[24] at the crossing point. Note that this value is approximately 1500 feet above the cruising altitude of the second aircraft. By the TCAS protocol, this would configure an unsafe operation scenario. Note also that, as described in Subsection 4.2, if the maneuver is not aborted, the first aircraft will reach an altitude of 1200 feet below the cruising altitude of the second aircraft, which is, in this case, a safe altitude. This is because a minimum vertical separation of 2000 feet is specified when cruising at altitudes above 29000 feet, and this minimum is reduced to 1000 feet when cruising below 29000 feet. One alternative would be to use the HyTech tool and specify a final region where the vertical distance reached of the first aircraft, at the crossing point, was exactly 32000 feet, the minimum required for safety, and parameterize the horizontal separation required to reach that final region. This would yield the minimum safe value for that parameter.

In a final case study, the vertical separation between the two aircraft was also parameterized, at the moment when the "climb" event is issued, while the horizontal separation was kept at 4000 meters. The HyTech tool showed that the vertical separation can be at most 36310/37, or about 97 meters, for the "climb" event to be issued. The vertical separation at the crossing point was still an unsafe 9615 meters, or about 31546 feet.

When, the HyTech tool was used to compute the relationship between the horizontal separation, at the point where the "climb" event occurs, and the vertical distance reached by the first aircraft at the crossing point, it was unable to complete the computation, due to the presence of multiplication library overflow errors.

5 Conclusions

The air traffic control protocol is a critical, real-time and reactive system. In addition, a possible system malfunction may cause enormous damages. The development of such systems demands the application of a rigorous validation and verification procedure. This

[24] About 31545 feet.

work describes a step in this direction, using the software HyTech as (semi) automatic formal verification tool.

Hybrid automata is the mathematical approach used to construct the models for the various real system agents studied in this work. The hybrid automata formalism allows for such agents to manifest a continuous, dynamic behavior, regulated by the asynchronous occurrence of discrete events. An Air Traffic Management (ATM) system, when using the Traffic alert and Collision Avoidance System (TCAS) protocol, exhibit all these characteristics. The cruising aircraft comprise the cooperating continuous dynamic agents of the system. The discrete asynchronous events arise from the exchange of commands, issued by the TCAS protocol, when some participating aircraft engage in a collision avoiding maneuver.

The formal models developed in this work were used to synthesize some parameter values for the operational behavior of two aircraft, cruising in opposite directions. The aircraft were allowed to perform different maneuvers, under the guidance of the TCAS protocol. Using the hybrid automata models, built from the TCAS protocol description, the HyTech tool synthesized some critical values for the vertical distance, and the vertical and horizontal separation between the cruising aircraft. The values obtained indicated which distances offer a safe operational scenario and, in some cases, indicated tighter values for some of these parameters.

The HyTech tool also proved very easy to use, once the hybrid automata models were in place. Changing or adding new parametric variables was easily achieved from one case study to the next one. Describing new final regions, in order to capture different aspects of the system behavior, presented no difficulties either.

Although hindered, at times, when the complexity of the models reached the limits of the tool, most of the case studies were successfully planned and run on a typical desktop PC. More realistic cases studies, involving more aircraft and contemplating more complex maneuvers, could be attempted by adapting and extending the source code of the HyTech tool in order to make it run in larger and faster machines.

Related work, studying the safety of (parts of) a subway system, can be found in [11, 9, 10].

References

[1] Rajeev Alur, Thomas A. Henzinger, and Pei-Hsin Ho. Automatic symbolic verification of embedded systems. In *Proceedings of the 14th Annual IEEE Real-Time Systems Symposium*, pages 2–11. IEEE Computer Society Press, 1993.

[2] Johan Bengtsson, Kim G. Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. UPPAAL — a tool suite for automatic verification of real–time systems. Technical Report RS-96-58, BRICS, Aalborg University, DENMARK and Department of Computer Systems, Uppsala University, Sweden, December 1996.

[3] Adilson Luiz Bonifácio, Arnaldo Vieira Moura, João Batista Camargo Jr., and Jorge Rady Almeida Junior. Análise e Verificação de Segmentos de Via de uma Malha Metroviária. In *Proceedings of the II Workshop on Formal Methods*, pages 13–22, Florianópolis, Brazil, October 1999. (In Portuguese).

[4] Adilson Luiz Bonifácio, Arnaldo Vieira Moura, João Batista Camargo Jr., and Jorge Rady Almeida Junior. Análise, Verificação e Síntese de Segmentos de Via de uma Malha Metroviária. Technical Report 18, Computing Institute, University of Campinas, Campinas, Brazil, August 1999. (In Portuguese).

[5] Adilson Luiz Bonifácio, Arnaldo Vieira Moura, João Batista Camargo Jr., and Jorge Rady Almeida Junior. Formal Parameters Synthesis for Track Segments of the Subway Mesh. In *7th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems*, pages 263–272, Edinburgh, Scotland, 3-7, April 2000. IEEE Computer Society Press.

[6] Adilson Luiz Bonifácio, Arnaldo Vieira Moura, João Batista Camargo Jr., and Jorge Rady Almeida Junior. Formal Verification and Synthesis for an Air Traffic Management System. Technical Report 05, Computing Institute, University of Campinas, Campinas, Brazil, February 2000.

[7] E. W. Dijkstra. Guarded commands, nondeterminacy and formal derivation of programs. *CACM*, 18(8):453–457, August 1975.

[8] Kathryn T. Heimerman. Air traffic control modeling. National Academy Press, The MITRE Corporation, 1998. As appears in the book entitled Frontiers of Engineering 1997.

[9] Thomas A. Henzinger and Pei-Hsin Ho. HyTech: The cornell hybrid technology tool. *Workshop on Hybrid Systems and Autonomous Control*, October 1994.

[10] Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. HYTECH: the next generation. In *Proceedings of the 16th IEEE Real-Time Systems Symposium*, pages 56–65, Pisa, Italy, 5-7, December 1995. IEEE Computer Society Press.

[11] Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. A user guide to HyTech. In E. Brinksma, W.R. Cleaveland, K.G. Larsen, T. Margaria, and B. Steffen, editors,

TACAS 95: Proceedings of the First Workshop on Tools and Algorithms for the Construction and Analysis of Systems, volume 1019 of *Lecture Notes in Computer Science*, pages 41–71. Springer-Verlag, 1995.

[12] Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. HYTECH: A model checker for hybrid systems. In O. Grumberg, editor, *CAV'97: Proceedings of the Ninth International Conference on Computer-Aided Verification*, volume 1254 of *Lecture Notes in Computer Science*, pages 460–463. Springer-Verlag, 1997.

[13] Pei-Hsin Ho. *Automatic Analysis of Hybrid Systems.* PhD thesis, Cornell University, August 1995.

[14] John Law. Acas ii programme. ACAS Programme Manager, January 1999.

[15] John Lygeros and Nancy Lynch. On the formal verification of the tcas conflict resolution algorithms. Technical report, Laboratory of Computer Science, Massachusetts Institute of Technology, 545 Technology Square, Cambridge, MA 02139, 1995.

[16] Phil Scott. Technology and business: Self-control in the skies. *Scientific American*, pages 24–55, January 2000.

[17] Claire Tomlin, George J. Pappas, and Shankar Satry. Conflict resolution for air traffic management: a study in multi-agent hybrid systems. In *IEEE Conference on Decision and Control*, Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, Berkeley, CA 94720, 1997.

[18] Lee F. Winder and James K. Kuchar. Evaluation of vertical collision avoidance maneuvers for parallel approach. In *AIAA Guidance, Naviation, and Control Conference*, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, Crambridge, MA 02139, August 1998. Boston, MA.

Epílogo

Neste capítulo foram estudadas a verificação e a síntese de parâmetros para a operação segura de um sistema de controle de tráfego aéreo. Foram considerados, especificamente, apenas manobras verticais entre dois aviões.

A modelagem do sistema, usando o formalismo de autômatos híbridos, permitiu a determinação de intervalos de valores para certos parâmetros operacionais do sistema. Neste estudo, o objetivo central foi evitar colisões nas aproximações verticais entre dois aviões que competem por rotas num mesmo espaço aéreo. Com os parâmetros de operação ajustados, foi possível estabelecer que o sistema se manteve seguro em algumas situações. Em outras situações, o sistema já violava certos padrões de proximidade, no momento em que as rotas se cruzavam.

O próximo capítulo apresenta algumas considerações finais, as conclusões do trabalho e algumas extensões para trabalhos futuros.

Capítulo 7

Conclusões

A operação diária de uma malha metroviária é um processo crítico, reativo e de tempo real. O mesmo se pode dizer de qualquer sistema para controle de tráfego aéreo. Devido ao potencial de causar enormes e irreparáveis danos, estes sistemas demandam um rigoroso processo de validação. Este trabalho descreve um passo nesta direção. Como modelo formal, foi usado o formalismo de autômatos híbridos. Este formalismo é bem adequado para modelar sistemas que compreendem inúmeros agentes individuais, que manifestam um comportamento dinâmico contínuo, e que sejam regulados pela ocorrência assíncrona de eventos discretos.

Uma malha metroviária apresenta todas estas características híbridas. Os componentes contínuos são representados pelos componentes físicos da malha, e os eventos assíncronos discretos surgem da troca de mensagens, durante sessões de comunicação. Um controle de tráfego aéreo, usando o TCAS, exibe também todas estas características. Os aviões em cruzeiro ditam a dinâmica contínua do sistema. Os eventos assíncronos discretos decorrem das trocas de comandos, emitidos quando alguns aviões participantes envolvem-se numa manobra para evitar colisões, operando sob a vigilância dos componentes do TCAS.

Num primeiro estudo, o modelo formal garantiu a segurança da operação de uma parte de uma malha metroviária, representada por um pátio de manobras, situado no fim da via. Foram verificados vários aspectos do controle da movimentação de trens, e também do funcionamento das máquinas de chave. Modelos similares podem ser desenvolvidos para verificar, da mesma forma, outros aspectos do sistema.

Posteriormente, num segundo estudo, estes mesmos modelos foram usados para sintetizar parâmetros para alguns aspectos operacionais do segmento de malha metroviária estudado. Foram sintetizados alguns parâmetros que ditam a movimentação e o posicionamento relativo dos trens, e também foram sintetizados alguns parâmetros de tempo da movimentação e travamento das máquinas de chave.

Num último estudo, modelos formais foram desenvolvidos e usados para sintetizar al-

guns valores para parâmetros do comportamento operacional de dois aviões, que trafegam num mesmo espaço aéreo em direções opostas. Foi permitido aos aviões realizar diferentes manobras, sob a vigilância dos componentes do TCAS. Usando modelos, construídos a partir da descrição do TCAS e baseados em autômatos híbridos, foi possível sintetizar alguns valores operacionais críticos para o sistema de gerenciamento de tráfego aéreo, tais como valores de separação vertical e horizontal. Estes números indicam medidas que oferecem um cenário operacional seguro e, em alguns casos, indicam valores mais justos para algumas destas distâncias.

A ferramenta (semi) automática HyTech foi usada como suporte computacional na análise dos modelos desenvolvidos. O HyTech foi capaz de verificar e sintetizar vários parâmetros operacionais de cada um desses modelos. Apesar de, em alguns casos, prejudicados devido a insuficiência de recursos computacionais, todos os experimentos de verificação e síntese foram programados e executados com sucesso sobre um PC de 350MHz típico, com 320MB de memória.

Dos sistemas abordados no trabalho, tanto a malha metroviária quanto o gerenciamento de tráfego aéreo, apresentam um grande potencial para extensões em várias direções.

Um aspecto adicional da operação da malha metroviária, e que poderia ser analisado usando-se autômatos híbridos, é o problema da *sombra*, descrito no Capítulo 3. A idéia consiste em modelar a estratégia de controle de velocidade dos trens, de forma a garantir a operação segura do sistema. Essa estratégia determinaria os códigos de velocidade padrão que seriam distribuídos ao longo da via e, também, os códigos de velocidade que seriam atribuídos aos circuitos de via, de acordo com a posição dos trens na via. Seria interessante sintetizar, ainda, alguns dos parâmetros utilizados nesse controle de velocidades, determinando melhores ajustes para os perfis de velocidade, de forma a melhorar a eficiência na operação do sistema, enquanto ainda observando todas as condições de segurança.

Outro aspecto que poderia ser modelado e analisado, ainda no caso da malha metroviária, é o *headway* do sistema. O *headway* é um parâmetro crítico na operação dos trens na via. Esse parâmetro estabelece uma distância mínima entre trens consecutivos que trafegam numa mesma linha. Esta distância mínima entre trens, se observada, deve ser suficiente para que a segurança do sistema permaneça garantida. É claro que essa distância mínima é uma função da velocidade relativa dos trens na via, e dos seus respectivos perfis de frenagem.

Outro estudo que poderia ser realizado na malha metroviária diz respeito à utilização de circuitos de via de comprimento variável. A ocupação de um circuito de via estaria, agora, sujeita à situação de tráfego na via. Com baixo tráfego na via a extensão do circuito de via poderia aumentar. No caso oposto, em situação de tráfego mais elevado, o circuito de via poderia encurtar, melhorando o tráfego naquela região. A idéia é que o desempenho e a eficiência do sistema aumente, já que haverá um melhor aproveitamento da via.

Há ainda outras características do sistema metroviário que poderiam ser modeladas. Uma dessas características diz respeito à movimentação dos trens nas estações metroviárias. Os trens devem operar de forma segura dentro das estações, como por exemplo, quando da abertura das portas automáticas para embarque e desembarque de passageiros. Existe também uma preocupação quanto à movimentação das portas dos trens nos locais corretos das estações, ou seja, as portas só devem se movimentar nos pontos demarcados para embarque e desembarque. Outra preocupação importante diz respeito à movimentação das portas do lado correto do trem. Como os trens trafegam em ambos os sentidos da via é possível, em um determinado momento, que operem de um lado da estação e, em outro momento, que trafeguem do lado oposto da estação.

Quanto ao sistema de gerenciamento de tráfego aéreo, inúmeros outros aspectos merecem consideração. Estudos de caso mais realistas, envolvendo mais aviões e manobras mais sofisticadas, poderiam ser realizados. Entre outros aspectos, o TCAS poderia ser estudado quando operando em situações mais complexas. Em particular, o TCAS poderia estar alerta não só para com aviões que trafegam na direção contrária, mas também poderia considerar aviões que viajam no mesmo sentido, em camadas superiores ou inferiores do espaço aéreo que se encontra.

Outro aspecto que poderia ser tratado em trabalhos futuros sobre o sistema de gerenciamento de tráfego aéreo, está ligado à verificação de propriedades de separações horizontais. Nesse trabalho, somente as distâncias e separações verticais foram estudadas. No caso de se considerar também parâmetros horizontais, outros aspectos das manobras seriam abordados, tais como ângulos e velocidades de ataque, próprios de conflitos horizontais com aviões em rotas paralelas.

Nos estudos encaminhados neste trabalho, o TCAS operou com tempos e intervalos mais longos. Seria interessante obter resultados com tempos e intervalos mais justos, aproximando o sistema modelado de uma concepção mais realista. Porém, como já citado, a ferramenta HyTech, nos estudos realizados, operou muito próxima de seus limites. Para se realizar estudos mais complexos, ou mais realistas, seria imprescindível, inicialmente, estender seu código e portá-lo para plataformas com maior capacidade computacional. Outro caminho seria a pesquisa de novos algoritmos para contornar tais problemas.

Bibliografia

[1] R. Alur, T. Henzinger, e H. Wong-Toi. Symbolic analysis of hybrid systems. Em *Proceedings of the 36th IEEE Conference on Decision and Control*, pp. 702–707, 1997. Invited survey.

[2] Rajeev Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, Pei-Hsin Ho, X. Nicollin, A. Olivero, J. Sifakis, e S. Yovine. The algorithmic analysis of hybrid systems. Em G. Cohen e J.-P. Quadrat, editores, *Proceedings of the 11th International Conference on Analysis and Optimization of Discrete Event Systems*, volume 199 de *Lecture Notes in Control and Information Science*, pp. 331–351. Springer-Verlag, 1994.

[3] Rajeev Alur, Thomas A. Henzinger, e Pei-Hsin Ho. Automatic symbolic verification of embedded systems. Em *Proceedings of the 14th Annual IEEE Real-Time Systems Symposium*, pp. 2–11. IEEE Computer Society Press, 1993.

[4] J. Bengtsson, W. O. D. Griffioen, K. J. Kristoffersen, K. G. Larsen, F. Larsson, P. Pettersson, e W. Yi. Verification of an audio protocol with bus collision using UPPAAL. Em Rajeev Alur e Thomas A. Henzinger, editores, *Proceedings of the Eighth International Conference on Computer Aided Verification (CAV)*, volume 1102 de *Lecture Notes in Computer Science*, pp. 244–256, New Brunswick, Nova Jersey, julho/agosto de 1996. Springer-Verlag.

[5] J. Bengtsson, K. G. Larsen, F. Larsson, P. Pettersson, e Wang Yi. UPPAAL in 1995. *Lecture Notes in Computer Science*, 1055:111–114, 1996.

[6] Johan Bengtsson, Kim Larsen, Fredrik Larsson, Paul Pettersson, Wang Yi, e Carsten Weise. New generation UPPAAL. Relatório técnico, BRICS, Dept. of Computer Science, Aalborg University, Dinamarca e Department of Computer Systems, Uppsala University, Suécia. Esse relatório técnico pode ser encontrado no endereço http://www.docs.uu.se/docs/rtmv/uppaal.

[7] Johan Bengtsson, Kim G. Larsen, Fredrik Larsson, Paul Pettersson, e Wang Yi. UPPAAL — a tool suite for automatic verification of real–time systems. Relatório Técnico

RS-96-58, BRICS, Aalborg University, Dinamarca e Department of Computer Systems, Uppsala University, Suécia, dezembro de 1996.

[8] Adilson Luiz Bonifácio e Itana Maria de Souza Gimenes. Estudo e comparação de provadores automáticos de teoremas. *Revista Tecnológica*, (7):75–85, outubro de 1998. Departamento de Ciência da Computação, Universidade Estadual de Maringá, Maringá.

[9] Adilson Luiz Bonifácio, Arnaldo Vieira Moura, João Batista Camargo Jr., e Jorge Rady Almeida Junior. Análise e Verificação de Segmentos de Via de uma Malha Metroviária. Em *Proceedings of the II Workshop on Formal Methods*, pp. 13–22, Florianópolis, outubro de 1999. (In Portuguese).

[10] Adilson Luiz Bonifácio, Arnaldo Vieira Moura, João Batista Camargo Jr., e Jorge Rady Almeida Junior. Análise, Verificação e Síntese de Segmentos de Via de uma Malha Metroviária. Relatório Técnico 18, Instituto de Computação, Universidade de Campinas, Campinas, agosto de 1999. (In Portuguese).

[11] Adilson Luiz Bonifácio, Arnaldo Vieira Moura, João Batista Camargo Jr., e Jorge Rady Almeida Junior. Formal Parameters Synthesis for Track Segments of the Subway Mesh. Em *7th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems*, pp. 263–272, Edinburgh, Escócia, 3-7, abril de 2000. IEEE Computer Society Press.

[12] Adilson Luiz Bonifácio, Arnaldo Vieira Moura, João Batista Camargo Jr., e Jorge Rady Almeida Junior. Formal Verification and Synthesis for an Air Traffic Management System. Relatório Técnico 05, Computing Institute, University of Campinas, Campinas, Brazil, fevereiro de 2000.

[13] Adilson Luiz Bonifácio, Arnaldo Vieira Moura, João Batista Camargo Jr., e Jorge Rady Almeida Junior. Formal Verification and Synthesis for an Air Traffic Management System. Em *3rd International Conference on Formal Methods in Computer Aided Design*, Austin, Texas, 1-3, novembro de 2000. Lecture Notes in Computer Science. (Submitted).

[14] João Batista Camargo Junior e Jorge Rady Almeida Júnior. Safety analysis case in the São Paulo Metro. Relatório técnico, Departamento de Sistams Digitais e Engenharia da Computação, Escola Politécnica - Universidade de São Paulo, São Paulo, 1998.

[15] Brenda D. Carpenter e James K. Kuchar. Probability-based collision alerting logic for closely-spaced parallel approach. Em *35th Aerospace Sciences Meeting & Exhibit*,

Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, Crambridge, Massachusetts 02139, janeiro de 1997. Reno, Nevada.

[16] P. R. D'Argenio, J.-P. Katoen, T. C. Ruys, e J. Tretmans. Modeling and verifying a bounded retransmission protocol. Relatório técnico, Faculty of Computing Science, University of Twente, AE Enschede, Holanda. Esse relatório técnico pode ser encontrado no endereço URL http://www.docs.uu.se/docs/rtmv/uppaal.

[17] P. R. D'Argenio, J.-P. Katoen, T. C. Ruys, e J. Tretmans. The bounded retransmission protocol must be on time! Em E. Brinksma, editor, *Proceedings of the Third Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, volume 1217 de *Lecture Notes in Computer Science*, pp. 416–431, Enschede, Holanda, abril de 1997. Springer-Verlag.

[18] E. W. Dijkstra. Guarded commands, nondeterminacy and formal derivation of programs. *CACM*, 18(8):453–457, agosto de 1975.

[19] José Henrique Zaccardi de Freitas, Antonio Accurso, e Ivaldo Lopes Mathias. A evolução tecnológica da cmsp e o estado da arte de sistemas de sinalização baseado em comunicação. Em *Revista Engenharia*, volume 529, pp. 116–124, 1998. (In Portuguese).

[20] N. Halbwachs, Y.-E. Proy, e Raymond P. Verification of linear hybrid systems by means of convex approximations. Em *International Symposium on Static Analysis*, volume LNCS 864 de *Lecture Notes in Computer Science*, 1994.

[21] Klaus Havelund, Arne Skou, Kim Guldstrand Larsen, e Kristian Lund. Formal modeling and analysis of an audio/video protocol: An industrial case study using UPPAAL. Relatório Técnico RS-97-31, BRICS, Aalborg University, Dinamarca e Bang & Olufsen, Dinamarca, novembro de 1997.

[22] Kathryn T. Heimerman. Air traffic control modeling. National Academy Press, The MITRE Corporation, 1998. Como aparece no livro intitulado Frontiers of Engineering 1997.

[23] Thomas A. Henzinger. The theory of hybrid automata. Em *Proceedings, 11th Annual IEEE Symposium on Logic in Computer Science*, pp. 278–292, New Brunswick, New Jersey, 27-30, julho de 1996.

[24] Thomas A. Henzinger e Pei-Hsin Ho. HyTech: The cornell hybrid technology tool. *Workshop on Hybrid Systems and Autonomous Control*, outubro de 1994.

[25] Thomas A. Henzinger, Pei-Hsin Ho, e Howard Wong-Toi. HYTECH: the next generation. Em *Proceedings of the 16th IEEE Real-Time Systems Symposium*, pp. 56–65, Pisa, Itália, 5-7, dezembro de 1995. IEEE Computer Society Press.

[26] Thomas A. Henzinger, Pei-Hsin Ho, e Howard Wong-Toi. A user guide to HyTech. Em E. Brinksma, W.R. Cleaveland, K.G. Larsen, T. Margaria, e B. Steffen, editores, *TACAS 95: Proceedings of the First Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, volume 1019 de *Lecture Notes in Computer Science*, pp. 41–71. Springer-Verlag, 1995.

[27] Thomas A. Henzinger, Pei-Hsin Ho, e Howard Wong-Toi. HYTECH: A model checker for hybrid systems. Em O. Grumberg, editor, *CAV'97: Proceedings of the Ninth International Conference on Computer-Aided Verification*, volume 1254 de *Lecture Notes in Computer Science*, pp. 460–463. Springer-Verlag, 1997.

[28] Thomas A. Henzinger e Howard Wong-Toi. Using HyTech to synthesize control parameters for a steam boiler. Em J.-R. Abrial, E. Börger, e H. Langmaack, editores, *Formal Methods for Industrial Applications: Specifying and Programming the Steam Boiler Control*, volume 1165 de *Lecture Notes in Control and Information Science*, pp. 265–282. Springer-Verlag, 1996.

[29] Pei-Hsin Ho. *Automatic Analysis of Hybrid Systems.* Tese de Doutorado, Cornell University, agosto de 1995.

[30] Pei-Hsin Ho e Howard Wong-Toi. Automated analysis of an audio control protocol. Em P. Wolper, editor, *Proceedings of the 7th International Conference On Computer Aided Verification*, volume 939 de *Lecture Notes in Computer Science*, pp. 381–394, Liege, Belgica, julho de 1995. Springer-Verlag.

[31] Per Stoffer Jensen, Thomas Mark Sorensen, Jesper Gravgaard, e Palle Klaerke Christensen. *Using AUTOGRAPH to Create Input for HyTech*, setembro de 1996. Esse manual do usuário para o AUTOGRAPH pode ser encontrado no endereço http://www-cad.eecs.berkeley.edu/~tah/hytech/atg_sun/.

[32] Kim G. Larsen, Paul Pettersson, e Wang Yi. Diagnostic model-checking for real-time systems. Relatório Técnico RS-96-57, BRICS, Aalborg University, Dinamarca e Department of Computer Systems, Uppsala University, Suécia, dezembro de 1996.

[33] Kim G. Larsen, Paul Pettersson, e Wang Yi. UPPAAL in a NUTSHELL. *International Journal on Software Tools for Technology Transfer*, 1(1):134–152, dezembro de 1997.

[34] John Law. Acas ii programme. ACAS Programme Manager, janeiro de 1999.

[35] M. Lindahl, P. Pettersson, e W. Yi. Formal design and analysis of a gear controller: an industrial case study using UPPAAL. Relatório técnico, Mecel AB, Göteborg, Suécia e Department of Computer Systems, Uppsala University, Suécia. Esse relatório técnico pode ser encontrado no endereço http://www.docs.uu.se/docs/rtmv/uppaal.

[36] Henrik Lönn e P. Pettersson. Formal verification of a TDMA protocol start-up mechnism. Relatório técnico, Department. of Computer Engineering, Chalmers University of Technology, Gothenburg, Suécia e Department of Computer Systems, Uppsala University, Uppsala, Suécia. Esse relatório técnico pode ser encontrado no endereço http://www.docs.uu.se/docs/rtmv/uppaal.

[37] John Lygeros e Nancy Lynch. On the formal verification of the tcas conflict resolution algorithms. Relatório técnico, Laboratory of Computer Science, Massachusetts Institute of Technology, 545 Technology Square, Cambridge, Massachusetts 02139, 1995.

[38] Nancy A. Lynch. *Distributed Algorithms.* Morgan Kaufmann, 1996.

[39] Donald Sannella. A survey of formal software development methods. Relatório Técnico ECS-LFCS-88-56, Laboratory for Foundations of Computer Science, Department of Computer Science, University of Edinburgh, julho de 1988.

[40] Phil Scott. Technology and business: Self-control in the skies. *Scientific American*, pp. 24–55, janeiro de 2000.

[41] Claire Tomlin, George J. Pappas, e Shankar Satry. Conflict resolution for air traffic management: a study in multi-agent hybrid systems. Em *IEEE Conference on Decision and Control*, Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, Berkeley, CA 94720, 1997.

[42] Lee F. Winder e James K. Kuchar. Evaluation of vertical collision avoidance maneuvers for parallel approach. Em *AIAA Guidance, Naviation, and Control Conference*, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, Crambridge, MA 02139, agosto de 1998. Boston, Massachusetts.

[43] Wang Yi e Mats Daniels. Automatic verification of real time communicating systems by constraint-solving. Relatório técnico, Department of Computer Systems, Uppsala University, Uppsala, Suécia. Esse relatório técnico pode ser encontrado no endereço http://www.docs.uu.se/docs/rtmv/uppaal.

Apêndice A

Segmento de Malha Metroviária

1.1 Os Autômatos do Modelo

```
var aux1, aux2, -- distância do trem
    t3, t4      -- tempo de movimentação do AMV
     : analog;
    k, w        -- controle da ocupação dos circuitos
     : discrete;

-- ------------------------------------------

automaton Ctrl_Trem_1
synclabs: ocupa1_1, ocupa1_2, ocupa1_3, ocupa1_4, Fechar_AMV3_t1,
          Abrir_AMV3_t1, desocupa1_2, desocupa1_5, desocupa1_4,
          desocupa1_3, desocupa1_1, Fechar_AMV4_t1, Abrir_AMV4_t1,
          ocupa1_5;
initially Ocupar;
loc Perfil_1: while aux1<=200 wait { daux1 in [7,9] }
        when aux1=200 goto Ocupar;
loc Perfil_2: while aux1>=0 wait { daux1 in [-9,-7] }
        when aux1=0 goto Ocupar;
loc Ocupando_1: while True wait { daux1=0 }
        when w<=2 goto Perfil_1;
        when w>=4 goto Perfil_1;
loc Ocupando_2: while True wait { daux1=0 }
        when w<=6 & w>=5 goto Perfil_2;
        when w<=3 goto Perfil_2;
```

```
        when w>=9 goto Perfil_2;
loc Ocupando_3: while True wait { daux1=0 }
        when k=9 & w>=5 & w<=6 sync Fechar_AMV3_t1
             do { aux1'=200 } goto Perfil_2;
        when k=3 & w=10 sync Abrir_AMV3_t1 do { aux1'=0 } goto Perfil_1;
        when k=3 & w=2 sync Abrir_AMV3_t1 do { aux1'=0 } goto Perfil_1;
        when k=9 & w<=2 sync Fechar_AMV3_t1 do { aux1'=200 } goto Perfil_2;
loc Ocupando_4: while True wait { daux1=0 }
        when k=8 & w>=5 & w<=6 sync Abrir_AMV4_t1
             do { aux1'=200 } goto Perfil_2;
        when k=8 & w=10 sync Abrir_AMV4_t1 do { aux1'=200 } goto Perfil_2;
        when k=7 & w=10 sync Fechar_AMV4_t1 do { aux1'=200 } goto Perfil_2;
        when k=4 & w=10 sync Fechar_AMV4_t1 do { aux1'=0 } goto Perfil_1;
        when k=8 & w=1 sync Abrir_AMV4_t1 do { aux1'=200 } goto Perfil_2;
        when k=4 & w<=2 sync Fechar_AMV4_t1 do { aux1'=0 } goto Perfil_1;
        when k=7 & w<=2 sync Fechar_AMV4_t1 do { aux1'=200 } goto Perfil_2;
loc Ocupar: while True wait { daux1=0 }
        when k=4 & asap sync desocupa1_4 do { k'=5 } goto Ocupar;
        when k=3 & asap sync desocupa1_3 do { k'=4 } goto Ocupar;
        when k=6 sync ocupa1_5 do { aux1'=200 } goto Ocupando_5;
        when k=5 sync ocupa1_5 do { aux1'=0 } goto Ocupando_5;
        when k=7 & asap sync desocupa1_4 do { k'=9 } goto Ocupar;
        when k=8 sync ocupa1_4 goto Ocupando_4;
        when k=1 & asap sync desocupa1_1 do { k'=3 } goto Ocupar;
        when k=9 & asap sync desocupa1_3 do { k'<=2 } goto Ocupar;
        when k=8 & asap sync desocupa1_4 do { k'=9 } goto Ocupar;
        when k=6 & asap sync desocupa1_5 do { k'=7 } goto Ocupar;
        when k=5 & asap sync desocupa1_5 do { k'=6 } goto Ocupar;
        when k=2 & asap sync desocupa1_2 do { k'=8 } goto Ocupar;
        when k=7 sync ocupa1_4 goto Ocupando_4;
        when k=9 sync ocupa1_3 goto Ocupando_3;
        when k=4 sync ocupa1_4 goto Ocupando_4;
        when k=3 sync ocupa1_3 goto Ocupando_3;
        when k=2 sync ocupa1_2 do { aux1'=200 } goto Ocupando_2;
        when k=1 sync ocupa1_1 do { aux1'=0 } goto Ocupando_1;
loc Ocupando_5: while True wait { daux1=0 }
        when k=6 & w<=2 do { aux1'=200 } goto Perfil_2;
        when k=5 & w<=2 do { aux1'=0 } goto Perfil_1;
        when k=5 & w>=8 do { aux1'=0 } goto Perfil_1;
        when k=6 & w>=8 do { aux1'=200 } goto Perfil_2;
end -- Ctrl_Trem_1
```

```
automaton Ctrl_Trem_2
synclabs: Abrir_AMV4_t2, Fechar_AMV4_t2, desocupa2_4, ocupa2_4,
          desocupa2_1, desocupa2_5, desocupa2_2, ocupa2_3,
          ocupa2_2, ocupa2_1, Abrir_AMV3_t2, Fechar_AMV3_t2,
          Perfil_1, desocupa2_3, ocupa2_5;
initially Ocupar;
loc Perfil_2: while aux2>=0 wait { daux2 in [-9,-7] }
        when aux2=0 goto Ocupar;
loc Ocupando_4: while True wait { daux2=0 }
        when w=8 & k>=5 & k<=6 sync Abrir_AMV4_t2
             do { aux2'=200 } goto Perfil_2;
        when w=8 & k=10 sync Abrir_AMV4_t2 do { aux2'=200 } goto Perfil_2;
        when w=7 & k=10 sync Fechar_AMV4_t2 do { aux2'=200 } goto Perfil_2;
        when w=4 & k=10 sync Fechar_AMV4_t2 do { aux2'=0 } goto Perfil_1;
        when w=7 & k<=2 sync Fechar_AMV4_t2 do { aux2'=200 } goto Perfil_2;
        when w=4 & k<=2 sync Fechar_AMV4_t2 do { aux2'=0 } goto Perfil_1;
        when w=8 & k=1 sync Abrir_AMV4_t2 do { aux2'=200 } goto Perfil_2;
loc Ocupar: while True wait { daux2=0 }
        when w=5 sync ocupa2_5 do { aux2'=0 } goto Ocupando_5;
        when w=6 sync ocupa2_5 do { aux2'=200 } goto Ocupando_5;
        when w=9 & asap sync desocupa2_3 do { w'<=2 } goto Ocupar;
        when w=4 & asap sync desocupa2_4 do { w'=5 } goto Ocupar;
        when w=3 & asap sync desocupa2_3 do { w'=4 } goto Ocupar;
        when w=1 sync ocupa2_1 do { aux2'=0 } goto Ocupando_1;
        when w=2 sync ocupa2_2 do { aux2'=200 } goto Ocupando_2;
        when w=3 sync ocupa2_3 goto Ocupando_3;
        when w=4 sync ocupa2_4 goto Ocupando_4;
        when w=9 sync ocupa2_3 goto Ocupando_3;
        when w=7 sync ocupa2_4 goto Ocupando_4;
        when w=2 & asap sync desocupa2_2 do { w'=8 } goto Ocupar;
        when w=5 & asap sync desocupa2_5 do { w'=6 } goto Ocupar;
        when w=6 & asap sync desocupa2_5 do { w'=7 } goto Ocupar;
        when w=8 & asap sync desocupa2_4 do { w'=9 } goto Ocupar;
        when w=1 & asap sync desocupa2_1 do { w'=3 } goto Ocupar;
        when w=8 sync ocupa2_4 goto Ocupando_4;
        when w=7 & asap sync desocupa2_4 do { w'=9 } goto Ocupar;

loc Ocupando_3: while True wait { daux2=0 }
        when w=9 & k>=5 & k<=6 sync Fechar_AMV3_t2
             do { aux2'=200 } goto Perfil_2;
        when w=3 & k=10 sync Abrir_AMV3_t2 do { aux2'=0 } goto Perfil_1;
```

```
        when w=9 & k<=2 sync Fechar_AMV3_t2 do { aux2'=200 } goto Perfil_2;
        when w=3 & k=2 sync Abrir_AMV3_t2 do { aux2'=0 } goto Perfil_1;
loc Perfil_1: while aux2<=200 wait { daux2 in [7,9] }
        when aux2=200 sync Perfil_1 goto Ocupar;
loc Ocupando_2: while True wait { daux2=0 }
        when k<=6 & k>=5 goto Perfil_2;
        when k<=3 goto Perfil_2;
        when k>=9 goto Perfil_2;
loc Ocupando_1: while True wait { daux2=0 }
        when k<=2 goto Perfil_1;
        when k>=4 goto Perfil_1;
loc Ocupando_5: while True wait { daux2=0 }
        when w=6 & k<=2 do { aux2'=200 } goto Perfil_2;
        when w=6 & k>=8 do { aux2'=200 } goto Perfil_2;
        when w=5 & k>=8 do { aux2'=0 } goto Perfil_1;
        when w=5 & k<=2 do { aux2'=0 } goto Perfil_1;
end -- Ctrl_Trem_2

automaton AMV_3
synclabs: Abrir_AMV3_t2, Abrir_AMV3_t1, Fechar_AMV3_t2, Fechar_AMV3_t1;
initially Normal;
loc Reverso: while True wait { dt3=0 }
        when True sync Fechar_AMV3_t1 do { t3'=0 } goto
        Movimentando_de_Reverso_para_Normal;
        when True sync Fechar_AMV3_t2 do { t3'=0 } goto
        Movimentando_de_Reverso_para_Normal;
        when True sync Abrir_AMV3_t1 goto Reverso;
        when True sync Abrir_AMV3_t2 goto Reverso;
loc Normal: while True wait { dt3=0 }
        when True sync Abrir_AMV3_t1 do { t3'=0 } goto
        Movimentando_de_Normal_para_Reverso;
        when True sync Abrir_AMV3_t2 do { t3'=0 } goto
        Movimentando_de_Normal_para_Reverso;
        when True sync Fechar_AMV3_t1 goto Normal;
        when True sync Fechar_AMV3_t2 goto Normal;
loc Movimentando_de_Normal_para_Reverso: while t3<=15 wait { dt3 in [4/5,1] }
        when t3=15 goto Reverso;
loc Movimentando_de_Reverso_para_Normal: while t3<=15 wait { dt3 in [4/5,1] }
        when t3=15 goto Normal;
end -- AMV_3

automaton AMV_4
```

```
synclabs: Abrir_AMV4_t2, Fechar_AMV4_t1, Fechar_AMV4_t2, Abrir_AMV4_t1;
initially Normal;
loc Reverso: while True wait { dt4=0 }
        when True sync Fechar_AMV4_t2 do { t4'=0 } goto
        Movimentando_de_Reverso_para_Normal;
        when True sync Fechar_AMV4_t1 do { t4'=0 } goto
        Movimentando_de_Reverso_para_Normal;
        when True sync Abrir_AMV4_t1 goto Reverso;
        when True sync Abrir_AMV4_t2 goto Reverso;
loc Normal: while True wait { dt4=0 }
        when True sync Abrir_AMV4_t1 do { t4'=0 } goto
        Movimentando_de_Normal_para_Reverso;
        when True sync Abrir_AMV4_t2 do { t4'=0 } goto
        Movimentando_de_Normal_para_Reverso;
        when True sync Fechar_AMV4_t2 goto Normal;
        when True sync Fechar_AMV4_t1 goto Normal;
loc Movimentando_de_Normal_para_Reverso: while t4<=15 wait { dt4 in [4/5,1] }
        when t4=15 goto Reverso;
loc Movimentando_de_Reverso_para_Normal: while t4<=15 wait { dt4 in [4/5,1] }
        when t4=15 goto Normal;
end -- AMV_4

automaton CV_1
synclabs: ocupa2_1, ocupa1_1, desocupa2_1, desocupa1_1;
initially Desocupa;
loc Desocupa: while True wait {  }
        when True sync ocupa1_1 goto Ocupa;
        when True sync ocupa2_1 goto Ocupa;
loc Ocupa: while True wait {  }
        when True sync desocupa1_1 goto Desocupa;
        when True sync desocupa2_1 goto Desocupa;
end -- CV_1

automaton CV_2
synclabs: ocupa2_2, ocupa1_2, desocupa2_2, desocupa1_2;
initially Desocupa;
loc Desocupa: while True wait {  }
        when True sync ocupa1_2 goto Ocupa;
        when True sync ocupa2_2 goto Ocupa;
loc Ocupa: while True wait {  }
        when True sync desocupa1_2 goto Desocupa;
```

```
        when True sync desocupa2_2 goto Desocupa;
end -- CV_2

automaton CV_3
synclabs: ocupa2_3, ocupa1_3, desocupa2_3, desocupa1_3;
initially Desocupa;
loc Desocupa: while True wait { }
        when True sync ocupa1_3 goto Ocupa;
        when True sync ocupa2_3 goto Ocupa;
loc Ocupa: while True wait { }
        when True sync desocupa1_3 goto Desocupa;
        when True sync desocupa2_3 goto Desocupa;
end -- CV_3

automaton CV_4
synclabs: ocupa2_4, ocupa1_4, desocupa2_4, desocupa1_4;
initially Desocupa;
loc Desocupa: while True wait { }
        when True sync ocupa1_4 goto Ocupa;
        when True sync ocupa2_4 goto Ocupa;
loc Ocupa: while True wait { }
        when True sync desocupa1_4 goto Desocupa;
        when True sync desocupa2_4 goto Desocupa;
end -- CV_4

automaton CV_5
synclabs: ocupa1_5, ocupa2_5, desocupa1_5, desocupa2_5;
initially Desocupa;
loc Desocupa: while True wait { }
        when True sync ocupa2_5 goto Ocupa;
        when True sync ocupa1_5 goto Ocupa;
loc Ocupa: while True wait { }
        when True sync desocupa2_5 goto Desocupa;
        when True sync desocupa1_5 goto Desocupa;
end -- CV_5
```

1.2 Verificação de Propriedades

1.2.1 Posicionamento entre Trens: Usando-se Predicados *Unsafe*

Análise: O trem no circuito cv_5 e outro trem no circuito cv_3 ou no circuito cv_4, no sentido padrão da via

```
var
   final_reg, init_reg, reached: region;
init_reg:=
     loc[Ctrl_Trem_1] = Ocupar &
     loc[Ctrl_Trem_2] = Ocupar &
     k=1 & w=1;
final_reg :=
      loc[Ctrl_Trem_1] = Perfil_1 &
      (loc[Ctrl_Trem_2] = Perfil_1 |
      loc[Ctrl_Trem_2] = Perfil_2) &
      k=5 &
      (w>=3 & w<=7);
reached:=
      reach forward from init_reg endreach;
if empty (reached & final_reg)
   then prints "Trem esta seguro";
   else prints "Trem violou a seguranca";
endif;
```

Resultado

```
Number of iterations required for reachability: 61
Trem esta seguro

================================================================
Max memory used =     0 pages =        0 bytes =   0.00 MB
Time spent      =     266.99u +     58.44s =     325.43 sec total
================================================================
```

1.2.2 Posicionamento entre Trens: Usando-se Predicados *Safe*

Análise: O trem no circuito cv_5 e outro trem no circuito cv_3 no sentido contrário da via

```
var
   final_reg, init_reg, reached: region;
init_reg:=
     loc[Ctrl_Trem_1] = Ocupar &
     loc[Ctrl_Trem_2] = Ocupar &
     k=1 & w=1;
final_reg :=
      loc[Ctrl_Trem_1] = Perfil_2 &
      (loc[Ctrl_Trem_2] = Perfil_1 |
       loc[Ctrl_Trem_2] = Perfil_2) &
      k=9 &
      (w=5 | w=6);
reached:=
      reach forward from init_reg endreach;
if empty (reached & final_reg)
   then prints "Trem esta seguro";
   else prints "Trem violou a seguranca";
endif;
```

Resultado

```
Number of iterations required for reachability: 61
Trem violou a seguranca

============================================================
Max memory used =     0 pages =        0 bytes =   0.00 MB
Time spent      =     214.01u +     53.82s =     267.83 sec total
============================================================
```

1.2.3 Posicionamento dos Trens em Relação aos AMVs: Usando-se Predicados *Unsafe*

Análise: O trem vindo do circuito cv_2 e o AMV_4 está em reverso após 180 metros

```
var
   final_reg, init_reg, reached: region;
init_reg:=
     loc[Ctrl_Trem_1] = Ocupar &
     loc[Ctrl_Trem_2] = Ocupar &
     loc[AMV_3] = Normal &
     loc[AMV_4] = Normal &
     k=1 & w=1 & t3=0 & t4=0;
final_reg :=
      loc[Ctrl_Trem_1] = Perfil_2 &
      k=8 & aux1<=20 &
      (loc[AMV_4] = Normal | loc[AMV_4] = Movimentando_de_Normal_para_Reverso
       | loc[AMV_4] = Movimentando_de_Reverso_para_Normal);
reached:=
      reach forward from init_reg endreach;
if empty(reached & final_reg)
   then prints "O trem passa com seguranca pelo AMV";
   else prints "O AMV pode nao estar na posicao correta";
endif;
```

Resultado

```
Number of iterations required for reachability: 82
O trem passa com seguranca pelo AMV

============================================================
Max memory used =      0 pages =         0 bytes =    0.00 MB
Time spent      =      126.90u +      33.05s =      159.95 sec total
============================================================
```

1.2.4 Posicionamento dos Trens em Relação aos AMVs: Usando-se Predicados *Safe*

Análise: O trem vindo do circuito cv_4 e o AMV_3 não está em normal após 160 metros

```
var
   final_reg, init_reg, reached: region;
init_reg:=
     loc[Ctrl_Trem_1] = Ocupar &
     loc[Ctrl_Trem_2] = Ocupar &
     loc[AMV_3] = Normal &
     loc[AMV_4] = Normal &
     k=1 & w=1 & t3=0 & t4=0;
final_reg :=
      loc[Ctrl_Trem_1] = Perfil_2 &
      k=9 & aux1<=40 &
      (loc[AMV_3] = Reverso | loc[AMV_3] = Movimentando_de_Normal_para_Reverso
       | loc[AMV_3] = Movimentando_de_Reverso_para_Normal);
reached:=
      reach forward from init_reg endreach;
if empty(reached & final_reg)
   then prints "O trem passa com seguranca pelo AMV";
   else prints "O AMV pode nao estar na posicao correta";
endif;
```

Resultado

```
Number of iterations required for reachability: 82
O AMV pode nao estar na posicao correta

============================================================
Max memory used =      0 pages =         0 bytes =    0.00 MB
Time spent      =      127.78u +      32.47s =      160.25 sec total
============================================================
```

1.3 Síntese de Parâmetros

1.3.1 Parametrizando a distância máxima segura do trem depois de sinalizar o AMV

Análise

```
var
 final_reg, init_reg, reached: region;
init_reg:=
     loc[Ctrl_Trem_1] = Ocupar &
     loc[AMV_3] = Normal &
     t3=0 & aux1=0;
final_reg :=
      loc[Ctrl_Trem_1] = Ocupar &
      aux1=dist_trem &
      (loc[AMV_3] = Normal | loc[AMV_3] = Movimentando_de_Normal_para_Reverso
       | loc[AMV_3] = Movimentando_de_Reverso_para_Normal);
reached:=
      reach forward from init_reg endreach;
print omit all locations
      hide non_parameters in
           reached & final_reg
      endhide;
```

Resultado

```
Number of iterations required for reachability: 10
     dist_trem >= 0   & 4dist_trem <= 675
```

1.3.2 Parametrizando a distância maxima percorrida do trem e o ponto de solicitação de movimentação do AMV

Análise

```
var
 final_reg, init_reg, reached: region;
```

```
init_reg:=
     loc[Ctrl_Trem_1] = Ocupar &
     loc[AMV_3] = Normal &
     t3=0 & aux1=pedido;
final_reg :=
      loc[Ctrl_Trem_1] = Ocupar &
      aux1=dist_trem &
      (loc[AMV_3] = Normal | loc[AMV_3] = Movimentando_de_Normal_para_Reverso
       | loc[AMV_3] = Movimentando_de_Reverso_para_Normal);
reached:=
      reach forward from init_reg endreach;
print omit all locations
      hide non_parameters in
           reached & final_reg
      endhide;
```

Resultado

```
Number of iterations required for reachability: 10
     pedido <= dist_trem   & 4dist_trem <= 4pedido + 675
```

1.3.3 Parametrizando o tempo de operação do AMV

Análise

```
var
   final_reg, init_reg, reached: region;
init_reg:=
     loc[Ctrl_Trem_1] = Ocupar &
     loc[AMV_3] = Normal &
     t3=0 & aux1=0;
final_reg :=
      loc[Ctrl_Trem_1] = Perfil_1 &
      aux1=200 &
      t3=tempo_amv &
      (loc[AMV_3] = Normal | loc[AMV_3] = Movimentando_de_Normal_para_Reverso
       | loc[AMV_3] = Movimentando_de_Reverso_para_Normal);
reached:=
```

```
      reach forward from init_reg endreach;
print omit all locations
      hide non_parameters in
           reached & final_reg
      endhide;
```

Resultado

```
Number of iterations required for reachability: 5
     9tempo_amv >= 160
```

1.3.4 Parametrizando o tempo de operação do AMV e ponto de sua solicitação de movimentação

Análise

```
var
   final_reg, init_reg, reached: region;
init_reg:=
     loc[Ctrl_Trem_1] = Ocupar &
     loc[AMV_3] = Normal &
     t3=0 & aux1=pedido;
final_reg :=
      loc[Ctrl_Trem_1] = Perfil_1 &
      aux1=200 &
      t3=tempo_amv &
      (loc[AMV_3] = Normal | loc[AMV_3] = Movimentando_de_Normal_para_Reverso
       | loc[AMV_3] = Movimentando_de_Reverso_para_Normal);
reached:=
      reach forward from init_reg endreach;
print omit all locations
      hide non_parameters in
           reached & final_reg
      endhide;
```

Resultado

```
Number of iterations required for reachability: 10
     pedido <= 200    & 45tempo_amv + 4pedido >= 800
```

1.3.5 Parametrizando a distância máxima segura do trem e o tempo de operação do AMV

Análise

```
var
   final_reg, init_reg, reached: region;
init_reg:=
     loc[Ctrl_Trem_1] = Ocupar &
     loc[AMV_3] = Normal &
     t3=0 & aux1=0;
final_reg :=
      loc[Ctrl_Trem_1] = Ocupar &
      aux1=dist_trem &
      t3=tempo_amv &
      (loc[AMV_3] = Normal | loc[AMV_3] = Movimentando_de_Normal_para_Reverso
       | loc[AMV_3] = Movimentando_de_Reverso_para_Normal);
reached:=
      reach forward from init_reg endreach;
print omit all locations
      hide non_parameters in
           reached & final_reg
      endhide;
```

Resultado

```
Number of iterations required for reachability: 10
     dist_trem >= 0    & 4dist_trem <= 45tempo_amv
```

1.3.6 Parametrizando a distância máxima segura do trem e o tempo de operação do AMV, bem como o ponto da solicitação de movimentação do AMV

Análise

```
var
   final_reg, init_reg, reached: region;
init_reg:=
     loc[Ctrl_Trem_1] = Ocupar &
     loc[AMV_3] = Normal &
     t3=0 & aux1=pedido;
final_reg :=
      loc[Ctrl_Trem_1] = Ocupar &
      aux1=dist_trem &
      t3=tempo_amv &
      (loc[AMV_3] = Normal | loc[AMV_3] = Movimentando_de_Normal_para_Reverso
       | loc[AMV_3] = Movimentando_de_Reverso_para_Normal);
reached:=
      reach forward from init_reg endreach;
print omit all locations
      hide non_parameters in
           reached & final_reg
      endhide;
```

Resultado

```
Number of iterations required for reachability: 10
     pedido <= dist_trem    & 4dist_trem <= 45tempo_amv + 4pedido
```

Apêndice B

Controle de Tráfego Aéreo

2.1 Os Autômatos do Sistema

```
var x1, x2, y1, y2   -- vertical and horizontal position of both aircraft
        : analog;
    k                -- it controls the taken climb
        : discrete;
    height, diff_height, diff_horiz   -- variable used to sinthesize
        : parameter;

-- ------------------------------------------

automaton Aircraft_1
synclabs: decrease, climb, reset, restart;
initially Cruise_B;
loc Cruise_B: while x1<=-4500 wait { dy1=0, dx1=280 }
when k=0 & x1=-4500 goto Descend;
loc Decrease: while True wait { dy1=-60, dx1=280 }
when x1=100 sync reset do { k'=1 } goto Cruise_A;
when True sync climb goto Climb;
loc Descend: while True wait { dy1=-50, dx1=280 }
when x1=100 sync reset do { k'=1 } goto Cruise_A;
when True sync climb goto Climb;
when True sync decrease goto Decrease;
loc Climb: while True wait { dy1=50, dx1=280 }
when x1=100 sync reset do { k'=1 } goto Cruise_A;
loc Cruise_A: while x1<=1000 wait { dy1=0, dx1=280 }
when True sync restart goto Cruise_B;
```

```
end -- Aircraft_1

automaton Aircraft_2
synclabs: reduce, reset, restart;
initially Cruise_B;
loc Cruise_B: while True wait { dy2=0, dx2=-280 }
when True sync reduce goto Reduced;
loc Reduced: while True wait { dy2=0, dx2=-250 }
when True sync reset goto Cruise_A;
loc Cruise_A: while True wait { dy2=0, dx2=-280 }
when True sync restart goto Cruise_B;
end -- Aircraft_2

automaton Controller
synclabs: decrease, climb, reduce, reset, restart;
initially Normal;
loc Normal: while x1<=x2-6000 wait {  }
when y1<y2+400 & x1=x2-7000 goto Descend;
when y1>=y2 + 300 & x1=x2-6000 sync climb do { k'=1 } goto Climb;
when y1>=y2+400 & x1=x2-7000 sync decrease goto Descend;
loc Descend: while x1<=1000 wait {  }
when True sync reset goto Reset;
when y1>=y2+300 & x1=x2-6000 sync reduce goto Reduced;
when y1>=y2+200 & x1=x2-5000 sync climb do { k'=2 } goto Climb;
loc Climb: while x1<=1000 wait {  }
when True sync reset goto Reset;
loc Reduced: while x1<=1000 wait {  }
when True sync reset goto Reset;
when y1>=y2+50 & x1=x2-4000 sync climb do { k'=3 } goto Climb;
loc Reset: while True wait {  }
when True sync restart goto Normal;
end -- Controller
```

2.2 Síntese de Parâmetros

2.2.1 A altura mínima antes da ação do controlador

Análise

```
var
   final_reg, init_reg, reached: region;
```

```
init_reg:=
     loc[Aircraft_1] = Cruise_B &
     loc[Aircraft_2] = Cruise_B &
     loc[Controler] = Normal &
     x1=-6000 & x2=6000 & y1=9750 & y2=9140 & k=0;
final_reg :=
     loc[Aircraft_1] = Descend &
     loc[Aircraft_2] = Cruise_B &
     loc[Controler] =  Normal &
     y1<=par;

reached:=
         reach forward from init_reg endreach;

print omit all locations
      hide non_parameters in
           reached & final_reg
      endhide;
```

Resultado

```
Will try hard to avoid library arithmetic overflow errors
Number of iterations required for reachability: 7
     7height >= 66375

Time spent = 0.63 sec total
```

2.2.2 No ponto crítico, com aumento da velocidade de descida

Análise

```
var
   final_reg, init_reg, reached: region;
init_reg:=
     loc[Aircraft_1] = Cruise_B &
     loc[Aircraft_2] = Cruise_B &
     loc[Controler] = Normal &
     x1=-6000 & x2=6000 & y1=9750 & y2=9140 & k=0;
final_reg :=
     loc[Aircraft_1] = Decrease &
     loc[Aircraft_2] = Cruise_B &
     loc[Controler] =  Descend &
```

```
     x1=x2 &
     y1<=par;

reached:=
        reach forward from init_reg endreach;

print omit all locations
      hide non_parameters in
           reached & final_reg
      endhide;
```

Resultado: o primeiro avião aumenta a descida

```
Will try hard to avoid library arithmetic overflow errors
Number of iterations required for reachability: 7
     7height >= 61750

Time spent = 0.63 sec total
```

Resultado: parametrizando a separação vertical e a distância vertical

```
Will try hard to avoid library arithmetic overflow errors
Number of iterations required for reachability: 4
     7diff_height = 3020   & 7height >= 61750

Time spent = 0.15 sec total
```

Resultado: parametrizando a separação horizontal e a distância vertical

```
Will try hard to avoid library arithmetic overflow errors
Number of iterations required for reachability: 4
     diff_horiz = 6648   & 7height >= 61794

Time spent = 0.15 sec total
```

2.2.3 Reduzindo a variação horizontal, sem subir

Análise

```
var
   final_reg, init_reg, reached: region;
init_reg:=
     loc[Aircraft_1] = Cruise_B &
```

```
     loc[Aircraft_2] = Cruise_B &
     loc[Controler] = Normal &
     x1=-6000 & x2=6000 & y1=9750 & y2=9140 & k=0;
final_reg :=
     loc[Aircraft_1] = Decrease &
     loc[Aircraft_2] = Reduced &
     loc[Controler] =  Reduced &
     x1=x2 &
     y1<=par;

reached:=
        reach forward from init_reg endreach;

print omit all locations
      hide non_parameters in
           reached & final_reg
      endhide;
```

Resultado: distância vertical enquanto desce

```
Will try hard to avoid library arithmetic overflow errors
Number of iterations required for reachability: 7
     371height >= 3259250

Time spent = 0.62 sec total
```

Resultado: distância vertical e separação vertical enquanto desce

```
Will try hard to avoid library arithmetic overflow errors
Number of iterations required for reachability: 7
     7diff_height <= 2270   & 371height >= 3259250

Time spent = 0.18 sec total
```

Resultado: distância vertical e separação horizontal enquanto desce

```
     Library arithmetic overflow errors in mutiplication occurs.
```

2.2.4 No ponto crítico e subindo

Análise

```
var
```

```
   final_reg, init_reg, reached: region;
init_reg:=
     loc[Aircraft_1] = Cruise_B &
     loc[Aircraft_2] = Cruise_B &
     loc[Controler] = Normal &
     x1=-6000 & x2=6000 & y1=9750 & y2=9140 & k=0;
final_reg :=
     loc[Aircraft_1] = Climb &
     loc[Aircraft_2] = Cruise_B &
     loc[Controler] =  Climb &
     x1=x2 &
     k=1 &
     y1<=par;

reached:=
         reach forward from init_reg endreach;

print omit all locations
      hide non_parameters in
           reached & final_reg
      endhide;
```

Resultado: distância vertical enquanto sobe

```
Will try hard to avoid library arithmetic overflow errors
Number of iterations required for reachability: 7
     7height >= 70125

Time spent = 0.66 sec total
```

Resultado: distância vertical e separação vertical enquanto sobe

```
Will try hard to avoid library arithmetic overflow errors
Number of iterations required for reachability: 7
     7diff_height <= 2395    & 7height >= 70125

Time spent = 0.18 sec total
```

Resultado: distância vertical e separação horizontal enquanto sobe

```
Will try hard to avoid library arithmetic overflow errors
Number of iterations required for reachability: 8
```

```
     28height >= 5diff_horiz + 250500
     & diff_horiz <= 9000    & diff_horiz >= 5528

Time spent = 0.21 sec total
```

2.2.5 Aumentar a descida, então subir

Análise

```
var
   final_reg, init_reg, reached: region;
init_reg:=
     loc[Aircraft_1] = Cruise_B &
     loc[Aircraft_2] = Cruise_B &
     loc[Controler] = Normal &
     x1=-6000 & x2=6000 & y1=9750 & y2=9140 & k=0;
final_reg :=
     loc[Aircraft_1] = Climb &
     loc[Aircraft_2] = Cruise_B &
     loc[Controler] =  Climb &
     x1=x2 &
     k=2 &
     y1<=par;

reached:=
         reach forward from init_reg endreach;

print omit all locations
      hide non_parameters in
           reached & final_reg
      endhide;
```

Resultado: distância vertical após aumentar a descida e subindo

```
Will try hard to avoid library arithmetic overflow errors
Number of iterations required for reachability: 7
     7height >= 68625

Time spent = 0.66 sec total
```

Resultado: distância vertical e separação vertical com descida aumentada, seguido por uma subida

```
Will try hard to avoid library arithmetic overflow errors
Number of iterations required for reachability: 7
     7diff_height <= 1520    & 7height >= 68625

Time spent = 0.18 sec total
```

Resultado: distância vertical e separação horizontal com descida aumentada, seguido por uma subida

```
Will try hard to avoid library arithmetic overflow errors
Number of iterations required for reachability: 7
     56height >= 11diff_horiz + 494000
     & diff_horiz >= 4840    & diff_horiz <= 7000

Time spent = 0.18 sec total
```

2.2.6 Abortando após aumentar a descida e reduzir a variação horizontal

Análise

```
var
   final_reg, init_reg, reached: region;
init_reg:=
     loc[Aircraft_1] = Cruise_B &
     loc[Aircraft_2] = Cruise_B &
     loc[Controler] = Normal &
     x1=-6000 & x2=6000 & y1=9750 & y2=9140 & k=0;
final_reg :=
     loc[Aircraft_1] = Climb &
     loc[Aircraft_2] = Reduced &
     loc[Controler] =  Climb &
     x1=x2 &
     k=3 &
     y1<=par;

reached:=
         reach forward from init_reg endreach;
```

```
print omit all locations
      hide non_parameters in
           reached & final_reg
      endhide;
```

Resultado: distância vertical após aumentar a descida, reduzindo e subindo

```
Will try hard to avoid library arithmetic overflow errors
Number of iterations required for reachability: 7
     371height >= 3567250

Time spent = 0.66 sec total
```

Resultado: distância vertical e separação vertical após aumentar a descida, reduzindo e subindo

```
Will try hard to avoid library arithmetic overflow errors
Number of iterations required for reachability: 7
     371diff_height <= 36310   & 371height >= 3567250

Time spent = 0.19 sec total
```

Resultado: distância vertical e separação horizontal após aumentar a descida, reduzindo e subindo

```
     Library arithmetic overflow errors in mutiplication occurs.
```