Um Enfoque Multiformalismos para Especificação de Sistemas de Segurança Crítica

Simone C. dos Santos (scs@di.ufpe.br)
Fabio Q. B. da Silva (fabio@di.ufpe.br)
Departamento de Informática
Universidade Federal de Pernambuco

Sumário

Chamamos de *sistemas de segurança crítica*, aqueles sistemas cuja falha pode causar danos ao ambiente ou perigo de vida aos seres humanos ou ambos. Tais sistemas requerem um alto grau de confiabilidade em seu desenvolvimento, antes de serem colocados em uso. Este artigo descreve a utilização de métodos formais para aumentar a confiabilidade no desenvolvimento de sistemas de segurança crítica. O enfoque é inovador no sentido em que propõe a utilização de formalismos distintos para descrever aspectos diferentes do sistema e produz a integração das várias especificações através de provas de consistência. Os resultados descritos aqui fazem parte do trabalho desenvolvido em [4].

Abstract

Safety-critical systems are those systems whose fault may cause environmental damage or endanger human life. Such systems require a high degree of reliability in its development, before being put to normal use. This article describes the use of formal methods to increase the reliability in the development of safety-critical systems. This approach is novel in the sense that it proposes the use of different formal methods to describe distinct aspects of the system and produces the integration of the various specifications via formal consistency proofs. The results reported here are part of the work developed in [4].

1 Introdução

Este artigo mostra como aplicar técnicas de engenharia de software e desenvolvimento formal de programas para aumentar a confiabilidade de sistemas de software cuja falha pode levar a catástrofes envolvendo danos materias ao ambiente, ou risco de vida ou ambos ([6], [11], [12]). Controladores de reatores nucleares e controladores de tráfego áereo são exemplos de tais *sistemas de segurança crítica*.

Para sistemas de segurança crítica, o engenheiro de software deveria ser capaz de garantir com total certeza que falhas não ocorrem. Entretanto, devido, principalmente, a impossibilidade da obtenção de todos os requisitos de segurança de tais sistemas, não se pode garantir confiabilidade total. Ao invés disso, procura-se conseguir um alto grau de confiança baseada em evidências suficientes para convencer o engenheiro de software e o cliente que o sistema é confiável dentro de uma margem de erro aceitável.

Uma operação essencial para produzir sistemas confiáveis é reduzir a complexidade e a quantidade de detalhes que devem ser considerados e tratados em uma dada fase do desenvolvimento. Duas propostas podem ser efetivamente utilizadas em busca desta operação: *abstração* e *decomposição*. Ou seja, por um lado estamos aptos a raciocinar em termos de *abstrações do problema real*, descritas em diversos níveis de detalhamento e destinadas a diferentes classes de leitores responsáveis por decisões sobre o sistema. Por outro lado, em sistemas suficientemente complexos, é praticamente impossível proceder diretamente dos conceitos iniciais aos programas executáveis. A divisão do desenvolvimento do sistema em subfases menores permite que cada parte possa ser tratada independentemente, reduzindo a quantidade de detalhes a serem considerados e, desta forma, proporcionando, em cada fase, maior entendimento do problema.

A utilização de métodos tradicionais de abstração e decomposição de sistemas é, certamente, uma poderosa ferramenta para aumentar a confiabilidade de sistemas complexos. Entretanto, sistemas de segurança crítica requerem um patamar muito mais elevado de confiabilidade: os métodos e técnicas utilizados na descrição do sistema devem resultar em especificações não ambíguas onde a validação (*estamos construindo o sistema desejado?*) possa ser realizada com alto grau de precisão e verificação (*estamos construindo o sistema corretamente?*) possa ser feita formalmente.

Este trabalho propõe a utilização de múltiplos métodos formais no desenvolvimento de software para sistemas de segurança crítica a fim de alcançar dois objetivos primordiais: aumentar o conhecimento sobre o funcionamento do sistema e com isso aumentar a precisão da validação; introduzir redundância no processo de desenvolvimento, diminuindo o risco de incorreções e, consequentemente, aumentando a confiabilidade da verificação.

A utilização de múltiplos métodos formais é justificada pela necessidade de tratar várias características diferentes, como comportamento determinístico, concorrente e temporal, que, em geral, não estão disponíveis em um único método formal. Outras propostas, como linguagens que integram mais de um formalismo ([7], [8]), também podem ser utilizadas com êxito no desenvolvimento de sistemas de segurança crítica, inclusive como um dos formalismos utilizados neste trabalho.

A aplicação de múltiplos paradigmas pode ser realizada, a princípio, de duas formas ortogonais. Primeiro, podemos aplicar um método diferente em cada fase do ciclo de desenvolvimento do sistema. Segundo, é possível utilizar vários métodos em uma mesma fase para especificar características complementares. Este trabalho utiliza o segundo enfoque em um estudo de caso de sistemas de segurança crítica, embora reconheça que tanto o primeiro enfoque como a combinação de ambos pode também ser vantajosa.

A utilização de vários métodos formais aqui sugerida concentra-se na fase de *análise de requisitos* do ciclo de desenvolvimento de software. A obtenção dos requisitos de segurança e

suas interrelações é uma fase fundamental na construção de sistema de segurança crítica. Erros, ambiguidades ou, simplesmente, omissões nesta fase podem levar à construção de sistemas que apesar de atenderem à especificação inicial, são inseguros por não considerarem alguns fatores de segurança ou considerá-los de forma incompleta ou ambígua.

Neste trabalho vamos apresentar duas especificações formais de um sistema, mostrando que, quando tomadas isoladamente, ambas são incompletas ao deixar de descrever importantes requisitos de segurança. Quando utilizadas em conjunto, as duas especificações fornecem uma descrição mais *completa* do sistema, aumentando a confiabilidade da análise de requisitos. Finalmente, estabelecemos que, quando utilizadas conjuntamente, estas especificações não são *conflitantes*, ou seja, que não existe contradição lógica entre as duas descrições. Desta forma, obtemos uma especificação consistente e mais completa.

Neste artigo, apresentaremos o enfoque discutido acima através de um estudo de caso. Para tornar mais clara a compreensão dos conceitos, vários detalhes formais foram omitidos. Em [4], o problema de utilização de vários formalismos e suas relações é discutido e desenvolvido de forma genérica. O leitor interessado nos detalhes técnicos e na abordagem mais teórica deste problema é convidado a ler a citação acima.

O trabalho é organizado em 6 seções. A Seção 2 descreve rápida e informalmente o sistema de caldeiras que será usado como estudo de caso. As Seções 3 e 4 apresentam fragmentos da especificação formal do sistema descritos em TLA (*Temporal Logic of Actions*, [9]) e Z ([14], [16]), considerando a contribuição destas especificações para o aumento da confiabilidade do sistema como um todo. A Seção 5 mostra como e por que é necessário relacionar as duas especificações e por fim, a Seção 6 apresenta as conclusões e considerações finais.

2 Um Sistema de Caldeiras

A descrição do sistema usado como exemplo neste artigo foi proposta no Simpósio Internacional de Projeto e Sistemas de Software Relacionados a Segurança (Wartloo, Canadá, [13]), com o objetivo de promover uma competição entre os participantes através de um problema genérico: um sistema de caldeiras (*Boiler System*).

O modelo do sistema consiste de um vasilhame aquecido, produzindo vapor d'água saturado através de um duto no topo do vasilhame. O fluxo de vapor varia segundo a variação da demanda externa (veja Figura 1).

O nível de água no vasilhame é medido por instrumentos, cuja leitura é afetada pela densidade da água, que é uma função da temperatura. Este nível deve estar acima do limite mínimo e abaixo do limite máximo permitido, evitando assim uma possível explosão proveniente do excesso de pressão dentro do boiler. Caso o nível de água ultrapasse o limite máximo, as bombas devem ser desligadas. No outro caso (nível abaixo do limite mínimo), as bombas devem ser ligadas.

O *Boiler System* é composto por:

- um vasilhame para aquecimento da água;
- quatro bombas;
- quatro monitores para as bombas;
- um dispositivo de medida do conteúdo de água;
- um dispositivo de medida para a taxa de vapor.

O vasilhame é alimentado por quatro bombas. Cada bomba possui um monitor que informa a taxa de fluxo de água bombeada por uma conexão separada das outras. O vasilhame possui um dispositivo de medida do nível da água em seu interior. A taxa de vapor que sai do vasilhame também é medida por um dispositivo.

Outro componente da especificação do *Boiler System* é o sistema de monitoração, responsável pela leitura de dados dos outros componentes e pela decisão de quando o sistema deve ser desli-

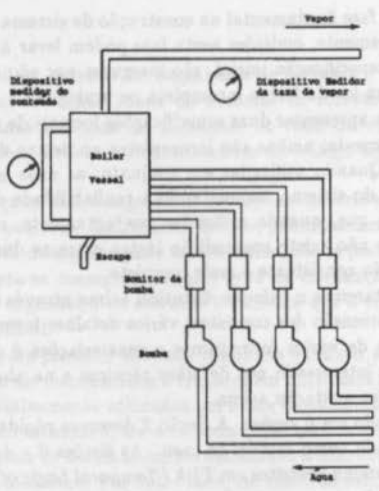

Figura 1: Boiler System

gado caso algum problema aconteça. Ele recebe medições de múltiplos sensores independentes, de diversas maneiras, pelo sistema de instrumentação. Portanto, a fusão de dados é necessária, para combinar os dados de diferentes fontes a fim de melhorar a estimativa e garantir a precisão e confiabilidade das informações. As bombas são ligadas e desligadas por um sistema de controle.

3 A Especificação em TLA

Utilizando uma estratégia de *comunicação de falhas*[1] e de *cálculo de nível*, as propriedades de segurança do *Boiler System* são descritas nesta seção usando a lógica TLA . Esta especificação foi desenvolvida em [2] e é utilizada aqui como ponto de partida para nossos estudos. Iniciamos com uma breve descrição de TLA.

3.1 Noções Básicas sobre TLA

A Lógica Temporal de Ações (TLA) de Lamport é a combinação de duas lógicas: uma lógica de ações e uma lógica temporal simples.

Fórmulas atômicas em TLA são *predicados* e *ações*. Um predicado pode ser visto como uma função de *estados* em valores *booleanos*, onde um estado é uma atribuição de valores a variáveis. Uma ação é uma expressão booleana formada de variáveis, variáveis decoradas e valores. Uma ação representa uma relação entre o estado anterior (representado pelas variáveis não decoradas) e o estado posterior (representado pelas variáveis decoradas). Assim, $x' = x + 1$ é válida entre os dois estados se o valor de x no estado posterior é uma unidade maior que o valor de x no estado anterior.

[1] Note que, *falhas* (do inglês, "*failure*"), neste contexto, dizem respeito a possíveis defeitos de dispositivos que, consequentemente, podem gerar *falhas* ("*fault*"), isto é, erros no sistema.

A sintaxe de TLA é:

$$\mathrm{F} ::= \mathrm{P} \mid \Box A \mid \neg \mathrm{F} \mid F_1 \wedge F_2 \mid \Box \mathrm{F}$$

onde P denota predicados e A ações.

Sistemas são sempre representados em TLA por fórmulas do tipo $\mathrm{P} \wedge \Box A$, onde P representa uma condição inicial. Para expressar que uma propriedade F é satisfeita por um sistema *Sys*, escreve-se $Sys \Rightarrow F$.

A notação $\bigwedge_{i \in I} . F_i$ é utilizada ao invés de $\forall i \in I . F_i$. Similarmente, $\bigvee_{i \in I} . F_i$, no lugar de $\exists i \in I . F_i$. Finalmente, $\sharp i . F_i$ significa o número de i's para o qual a fórmula F_i é válida.

3.2 Modelagem do Sistema em TLA

A especificação em TLA do *Boiler System* proposta em [2] usa consistência de dados para detectar falhas. O modelo do *Boiler System* em TLA captura os componentes físicos e o sistema de monitoração. O sistema de monitoração lê informações dos dispositivos e decide quando o sistema entra em *shutdown*. As bombas são ligadas pelo sistema de controle (que não faz parte do modelo). A comunicação entre os sistemas de instrumentação (responsável pelo envio de dados físicos), de monitoração e de controle, não é modelada.

As variáveis do Sistema L, S e P representam, respectivamente, o nível do conteúdo do vasilhame, taxa de vapor e fluxo das bombas. L_c e L_m, S_c e S_m, P_c e pm_i correspondem, respectivamente, aos valores calculados e medidos do nível, vapor e fluxo de água. O número de bombas operantes é representado por np e pi_i é indicador *on/off* para a bomba i. L_p, S_p e P_p representam os limites físicos dos dispositivos. *Safe*, é o limite de segurança do nível de água e K o fluxo de água por bomba. f_d e r_d significam falha real e falha comunicada do dispositivo d, respectivamente. c_D corresponde a consistência de leituras do conjunto de dispositivos D e finalmente up é a variável que indica se o sistema está operante ou em modo *shutdown*.

O índice inferior d das variáveis f_d e r_d variam sobre os elementos de $Dev \stackrel{\text{def}}{=} \{l, p, s\}$, onde l diz respeito ao nível da água, s ao fluxo de vapor e p às bombas.

O Sistema do Boiler é descrito na forma padrão de uma especificação em TLA:

$$BSys \stackrel{\text{def}}{=} Init \wedge \Box Step$$

onde a condição inicial *Init* estabelece que, inicialmente, conhecemos o nível real do boiler que deve estar dentro dos limites de segurança e *Step* representa os componentes do Sistema.

$$Init \stackrel{\text{def}}{=} L_c = L \wedge up \wedge L \subseteq Safe$$

O modelo do *Boiler System* possui cinco componentes: o comportamento do boiler, o comportamento em caso de *shutdown*, fusão de dados, consistência de dados e determinação do nível de água.

$$Step \stackrel{\text{def}}{=} Boiler \wedge Shutdown \wedge Fusion \wedge Cons \wedge Level$$

Aqui, apenas mostraremos a modelagem, através da especificação em TLA, dos três primeiros componentes citados acima, denominados por: *Boiler*, *Shutdown* e *Fusion*.

O modelo do componente *Boiler* diz respeito ao relacionamento entre as variáveis físicas do sistema, bem como considerações sobre falhas:

$$
\begin{aligned}
Boiler \stackrel{\text{def}}{=}\ & L' = L + (P' - S') \\
& \wedge P = K.np \\
& \wedge S \subseteq S_p \\
& \wedge P \subseteq P_p \\
& \wedge \neg f_l \Rightarrow L \subseteq L_m \\
& \wedge \neg f_s \Rightarrow S \subseteq S_m \\
& \wedge \neg f_p \Rightarrow np = \sharp i.pm_i \\
& \wedge \bigwedge_{d \in Dev} \left(f_d \Rightarrow \bigvee_{D \supseteq \{d\}} \neg c_D \right) \\
& \wedge \bigwedge_{D \subseteq Dev} \left(\neg c_D \Rightarrow \bigvee_{d \in D} f_d \right)
\end{aligned}
$$

As três primeiras considerações de falhas no modelo *Boiler* afirmam que um dispositivo *sem* falha informa seus valores dentro da precisão especificada. As duas últimas mostram que um dispositivo *com* falha informa valores inconsistentes e inconsistências surgem *unicamente* na presença de falhas dos dispositivos. Estas considerações são satisfatórias se não considerarmos controle e comunicação, uma vez que falhas podem se referir, por exemplo, a comunicação entre os sistemas de instrumentação e de monitoração. Para atingirmos níveis satisfatórios de segurança, estes sistemas devem ser especificados e as condições de falha relativas a eles devem ser levadas em consideração. No enfoque proposto aqui, estes sistemas podem ser descritos em formalismos adequados e posteriormente combinados através de provas de consistência.

O modelo *Shutdown* determina o valor da variável *up* que assume *true* quando o nível do boiler estiver dentro dos limites de segurança:

$$Shutdown \stackrel{\text{def}}{=} up = L_\epsilon \subseteq Safe$$

O modelo *Fusion* contém uma estratégia geral para detectar falhas de inconsistência entre valores medidos.

$$Fusion \stackrel{\text{def}}{=} \bigwedge_{d \in Dev} \left(r_d = \bigvee_{D \supseteq \{d\}} \neg c_D \right)$$

O modelo de *Fusion* descreve que, caso exista uma inconsistência em algum dispositivo medidor do nível, vapor ou fluxo de água, são emitidas falhas para todos os outros dispositivos. E mais, falhas são emitidas unicamente em função da consistência de dados entre fontes redundantes. Esta estratégia pessimista foi escolhida para simplificar o problema e, consequentemente, os aspectos que degradam o sistema aos poucos, quando, por exemplo, apenas um dispositivo deixar de funcionar corretamente, deixaram de ser considerados.

Em [2], várias propriedades matemáticas do sistema foram estabelecidas, indicando que o sistema é seguro de acordo com as hipóteses da modelagem.

3.3 Deficiências da Especificação em TLA

Identificamos alguns aspectos importantes que não estão descritos na especificação em TLA:

- Inconsistências além daquelas geradas por falhas dos dispositivos, por exemplo erros de comunicação (falhas de protocolo entre subsistemas) ou em cálculos de variáveis (erros algébricos do programa).
- Comunicação entre os sistemas de instrumentação, monitoração e controle.

• Identificação dos possíveis estados que o sistema pode assumir a medida que alguns dispositivos falham individualmente, como primeiro passo do projeto da arquitetura do sistema.

Para obtermos uma descrição formal mais completa do sistema, os itens acima devem ser considerados. No enfoque que propomos neste trabalho, utilizamos outros formalismos, mais adequados à descrição das características citadas, para complementar a especificação em TLA. Assim, o problema de cálculo incorreto do valor das variáveis pode ser especificado utilizando-se um método algébrico e a comunicação pode ser descrita em um cálculo de processos concorrentes. Estes aspectos não serão tratados aqui, mas estão parcialmente descritos em [4].

Analisando particularmente o terceiro ponto enfatizado acima, podemos imaginar o sistema modelado como uma sequência de transição de estados gerados por situações que comprometam a estabilidade do mesmo. Por exemplo, o sistema no seu estado *normal* assumiria o estado *shutdown* caso os limites de segurança fossem violados. Neste modelo, o tratamento dado às situações de falha varia de estado para estado. Por exemplo, caso o sistema esteja funcionando *normalmente* e o dispositivo medidor do nível de água apresente alguma falha, o sistema assumirá um estado de *emergência* e não necessariamente *shutdown*. Entretanto, caso esta falha aconteça quando o sistema estiver com outro dispositivo de medida (do vapor ou das bombas) com defeito, o funcionamento do sistema passa a ser inviável e, portanto, o sistema deve assumir o estado *shutdown*. Na seção seguinte, apresentamos esta visão do sistema através de uma especificação orientada a modelos utilizando Z ([14], [16]).

4 A Especificação em Z

Z é uma notação baseada em lógica de predicados e teoria de conjuntos para modelagem rigorosa e construtiva de sistemas. Uma especificação em Z é formada por vários tipos de declarações para introduzir nomes, expressões que denotam termos, predicados que expressam propriedades dos valores dos nomes declarados e esquemas que impõem uma estrutura sobre as declarações e predicados. Assumimos que o leitor seja familiarizado com esta notação, descrita detalhadamente em [8].

4.1 Modelagem do Sistema em Z

O diagrama de estados da Figura 2 foi extraído a partir da descrição informal do sistema de caldeiras proposto em [13]. Seis modos de operação foram identificados: **computer self test, system test and initialization, normal operation, degraded operation, emergency operation** e **shutdown**. Com base nestes modos de operação, descrevemos um modelo do sistema que pode ser visto operacionalmente como uma máquina de transição de estados. A máquina parte de um estado inicial (**computer self test**) para um novo estado através da execução de operações que devem atender a certos requisitos e devem produzir resultados.

No diagrama da Figura 2, o nome de cada transição é formado pelos dois estados envolvidos na transição, separados por hífem e no sentido da seta.

A especificação em Z descreve o comportamento do *Boiler System* baseada na transição de seis estados representados pelo tipo enumerado *STATE*. Apenas parte da especificação é aqui apresentada. Para maiores detalhes veja [4].

Por convenção, neste trabalho, um esquema é referenciado em letras estilo *itálico* e inicia-se com letra maiúscula. As varjáveis são escritas em letras minúsculas salvo aquelas cujos nomes são formados por uma única letra. Para evitar dúvidas quando falamos de uma transição de estados (representada por um esquema) e um estado específico, este último será escrito em **negrito**.

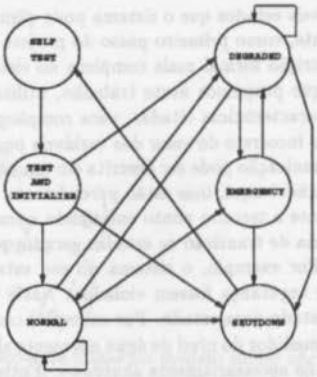

Figura 2: Diagrama de estados do *Boiler System*

As constantes *lim-sup* e *lim-inf* representam os limites físicos do nível de água no vasilhame que não poderá ultrapassar seu limite superior nem obter um nível menor que seu limite inferior. S_p e P_p representam os limites físicos da taxa de vapor máxima e fluxo total de água bombeada, respectivamente. A constante K é o fluxo de água por bomba e np, o número total de bombas no sistema (ligadas ou não).

$STATE ::= Self.Test \mid Test.And.Init \mid Normal \mid Degraded \mid Emergency \mid Shutdown$
$SITUATION := OPEN \mid CLOSED$
$INDICATOR := ON \mid OFF$

$lim\text{-}inf : \mathbf{R}$

$lim\text{-}sup : \mathbf{R}$

$S_p : \mathbf{R}$

$P_p : \mathbf{R}$

$K : \mathbf{R}$

$np : \mathbf{N}^+$

$np = 4$

O estado do *Boiler System* é representado pelo esquema *BSys* onde L, S, e P são respectivamente os valores reais do nível de água no vasilhame, da taxa de vapor e do fluxo de água bombeada. Assumimos que estes valores são constantemente alterados durante o funcionamento do sistema, embora as operações responsáveis por estas alterações não estejam representadas nesta especificação. L_c e S_c representam os valores do nível de água e taxa de vapor calculados pelo sistema de instrumentação. Estes valores calculados serão comparados com valores medidos a fim de se comprovar a consistência de dados. No caso das bombas em seu estado normal, o valor medido de cada uma deve ser igual a constante K.

Os componentes de estado *pumps*, *valve* e *up* indicam se as bombas estão ligadas, o escape está aberto e se o sistema está em funcionamento, respectivamente.

BSys

$L : \mathbf{R}$
$S : \mathbf{R}$
$P : \mathbf{R}$
$L_c : \mathbf{R}$
$S_c : \mathbf{R}$
$pumps : Seq\ INDICATOR$
$valve : SITUATION$
$up : INDICATOR$
$state : STATE$

$\sharp(pumps) = np$
$S \leq S_p$
$P \leq P_p$
$L_c = L + (P - S)$
$S_c \leq S_p$

A função *npump-ok* calcula o número de bombas operantes que estão funcionando corretamente.

$npump\text{-}ok : \quad Seq\ \mathbf{R} \rightarrow \mathbf{N}$

$npump\text{-}ok = \lambda p : Seq\ \mathbf{R} . \sharp \{ i : dom\ pumps \mid pumps(i) = ON \wedge p(i) = K \}$

O esquema *Bsys** foi criado para agrupar situações comuns a várias transições de estado representadas por esta especificação. O esquema *Limit-ok* verifica se os limites para o nível de água estão sendo respeitados. Os esquemas *Level-Ok*, *Steam-Ok* e *Pumps-Ok* verificam se os medidores estão funcionando corretamente através da comparação entre os valores calculados e medidos (mostrados nos monitores). Visto que, sistemas de segurança são frequentemente construídos sobre o princípio de diversidade e redundância, considerações de diversas ou redundantes fontes são de particular interesse.

O esquema *BSys-Off* representa o estado em que o sistema não está em funcionamento, ou seja, sistema desligado, bombas desligadas e o escape aberto.

Usaremos várias vezes ao longo da especificação em Z a negação do esquema *Pumps-Ok* significando que nem todas as bombas estão funcionando corretamente. Embora este esquema não esteja normalizado ($P_{mi}?$ tem o tipo $Seq\ \mathbf{R}$), assumimos aqui que a negação deste esquema atinge unicamente o predicado $npumps\text{-}ok(p_{mi}?) = np$.

*BSys**

$\Delta BSys$

$pumps = pumps'$
$valve = valve'$
$up = up'$

Limit-Ok

$L : \mathbf{R}$

$(L \leq lim\text{-}sup) \wedge (L \geq lim\text{-}inf)$

Level-Ok

$L_c : \mathbf{R}$
$L_m? : \mathbf{R}$

$L_c = L_m?$

Steam-Ok

$S_e : \mathbf{R}$

$S_m? : \mathbf{R}$

$S_e = S_m?$

Pumps-Ok

$P_{mi}? : Seq\ \mathbf{R}$

$\text{npumps-ok}(P_{mi}?) = np$

BSys-Off

$\Delta Bsys$

$ran\ pumps' = \{OFF\}$

$valve' = OPEN$

$up' = OFF$

O estado inicial do sistema é representado pelo esquema *Self.Test*, onde o sistema está desligado e os valores zerados. A transição dos estados **computer self test** para **system test and initialization** representa a operação em que o sistema é efetivamente ligado. Inicialmente os limites são respeitados e os dispositivos medidores estão em perfeito estado de funcionamento. O sistema e as bombas são ligadas (em um total de 4 bombas). Nem todas as transições estão especificadas neste artigo. Aqui, apenas descreveremos as transições *Test.And.Init-To-Normal* e *Normal-To-Shutdown*.

A passagem do estado de inicialização do sistema **system test and initialization** para o estado **normal** requer apenas que os dispositivos estejam funcionando bem, uma vez que esta transição representa o momento imediatamente após o sistema iniciar o seu funcionamento.

Test.And.Init-To-Normal

$BSys\ *$

$Limit\text{-}Ok$

$Level\text{-}Ok$

$Steam\text{-}Ok$

$Pumps\text{-}Ok$

$state = Test.And.Init$

$state' = Normal$

Estando no estado **normal** o sistema poderá assumir três situações. Caso o dispositivo medidor da taxa de vapor ou algum dos dispositivos medidores do fluxo de água não estiver funcionando corretamente ele passará para o estado **degraded**. Se o problema estiver com o dispositivo medidor do nível da água, passará para o estado **emergency**. A última situação diz respeito a ultrapassagem dos limites físicos do nível de água que poderá causar uma explosão, e consequentemente, o sistema passará para o estado **shutdown**. Apenas esta última transição é aqui apresentada.

Normal-To-Shutdown

$BSys\text{-}Off$

$\neg Limit\text{-}Ok$

$state = Normal$

$state' = Shutdown$

4.1.1 Propriedades da Especificação em Z

Uma propriedade importante a ser provada sobre a especificação em Z do *Boiler System* é mostrar que em um determinado momento o sistema estará em um *único estado* daqueles representados na Figura 2. Isto significa mostrar que todas transições do diagrama da Figura 2 foram especificadas e que não existe nenhuma situação tal que mais de uma transição satisfaça uma mesma pré-condição, eliminando não-determinismos. Assim, definimos o teorema abaixo, onde *Pre-OP* significa as pré-condições de todos os esquemas que representam as transições de estado do *Boiler System* e $Pre\text{-}OP_i$, as pré-condições dos esquemas que representam as transições para o estado i.

Teorema 1: *A disjunção das pré-condições das diversas operações é verdadeira e a conjução das pré-condições de cada par de operações é falsa, onde cada operação equivale a uma transição de estado.*

Lema 1 (Habilitação das Transições:

Seja $OP \stackrel{def}{=} Self.Test \vee Test.and.Init \vee Normal \vee Degraded \vee Emergency \vee Shutdown$
então $Pre\text{-}OP = [BSys \mid true]$

Lema 2 (Determinismo):

$\forall i, j\ Pre\text{-}OP_i \wedge Pre\text{-}OP_j = [BSys \mid false], 1 \leq i, j \leq n\ e\ i \neq j,$
onde n é igual ao número de estados em OP.

Neste artigo, apenas comentamos parte da prova dos Lemas 1 e 2, referente às transições que partem do estado **normal**. As provas são descritas inteiramente em [4].

Prova (Lema 1)

Observando o diagrama da Figura 2, identificamos quatro possíveis estados que podem ser alcançados a partir do estado **normal**: o próprio estado **normal** (caso em que o sistema não apresenta qualquer defeito) e os estados **degraded**, **emergency** e **shutdown**. Estas transições são especificadas pelos esquemas *Normal-To-Normal*, *Normal-To-Degraded*, *Normal-To-Emergency* e *Normal-To-Shutdown*. Assim, extraímos as pré-condições de cada um destes esquemas, agrupando-as em uma tabela (veja Tabela 1), como sugerido em [16]. Especialmente neste caso, tornou-se conveniente reescrever as pré-condições em uma nomemclatura auxiliar, diminuindo o tamanho das fórmulas contendo estas pré-condições e, com isso, facilitando a descrição e entendimento das provas.

Lim-Ok	$(L \leq lim\text{-}sup) \wedge (L \geq lim\text{-}inf)$
L-Ok	$L_c = L_m?$
S-Ok	$S_c = S_m?$
P-Ok	$npumps\text{-}ok(P_{mi}?) = np$

Esquemas	Pré-Condições
Normal-To-Normal	*Lim-Ok* $\wedge$ *L-Ok* $\wedge$ *S-Ok* $\wedge$ *P-Ok* $\wedge$ *state* = *Normal*
Normal-To-Degraded	*Lim-Ok* $\wedge$ *L-Ok* $\wedge$ ($\neg$ *S-Ok* $\vee \neg$ *P-Ok*) $\wedge$ *state* = *Normal*
Normal-To-Emergency	*Lim-Ok* $\wedge \neg$ *L-Ok* $\wedge$ *state* = *Normal*
Normal-To-Shutdown	$\neg$ *Lim-Ok* $\wedge$ *state* = *Normal*

Tabela 1: Pré-condições das transições do estado **normal**

Usando as pré-condições da Tabela 1 temos

(Lim-Ok ∧ *L-Ok* ∧ *S-Ok* ∧ *P-Ok* ∧ *state = Normal)* (1)
∨
(Lim-Ok ∧ *L-Ok* ∧ (¬ *S-Ok* ∨¬ *P-Ok)* ∧ *state=Normal)* (2)
∨
(Lim-Ok ∧¬ *L-Ok* ∧ *state = Normal)* (3)
∨
(¬ Lim-Ok ∧ *state = Normal)* (4)

Aplicando as regras de distribuição da disjunção da lógica de predicados de primeira ordem juntamente com a tautologia $P \vee (\neg P)$, conseguimos reduzir a fórmula acima para o termo *state = Normal*.

Primeiro, a partir de (1) ∨ (2) obtemos *Lim-Ok* ∧ *L-Ok* ∧ *state = Normal*, que chameremos de (1-2). Assim, reduzimos a fórmula (1) ∨ (2) ∨ (3) ∨ (4) ∨ para a fórmula (1-2) ∨ (3) ∨ (4). Da mesma forma, a partir de (1-2) ∨ (3), chegamos a *Lim-Ok* ∧ *state = Normal*, representada por (1-2-3). Por fim, partindo de (1-2-3) ∨ (4), obtemos *state = Normal*. Para maiores detalhes desta prova veja [4].

Aplicando o mesmo procedimento (desde a construção da tabela de pré-condições à organização das provas) para os estados **self test, test and init, degraded, emergency e shutdow**, obtemos a fórmula disjuntiva

state=Self.Test ∨ *state=Test.And.Init* ∨ *state=Normal* ∨
state=Degraded ∨ *state=Emergency* ∨ *state=Shutdown*

que, pela definição do tipo enumerado *STATE*, é igual a *true*, terminando a prova do Lema 1. □

Prova (Lema 2)

O Lema 2 exige que, ao tomarmos dois estados qualquer e_1 e e_2 dos seis estados anteriormente definidos, a conjunção entre as pré-condições necessárias para o sistema estar no estado e_1 e as pré-condições necessárias para o sistema estar no estado e_2 seja falsa. Ou seja, o sistema de transição é determinístico. Por exemplo, considerando os estados **normal** e **emergency**, identificamos as seguintes condições apresentadas na tabela abaixo.

Pré-Condições para o sistema no estado normal
Lim-Ok ∧ *L-Ok* ∧ *S-Ok* ∧ *P-Ok* ∧ *state = Test.and.Init*
Lim-Ok ∧ *L-Ok* ∧ *S-Ok* ∧ *P-Ok* ∧ *state = Normal*
Lim-Ok ∧ *L-Ok* ∧ *S-Ok* ∧ *P-Ok* ∧ *state = Degraded*
Pré-Condições para o sistema no estado emergency
Lim-Ok ∧¬ *L-Ok* ∧ *state = Normal*
Lim-Ok ∧¬ *L-Ok* ∧ *state = Degraded*

Assim, considerando as condições abaixo

(a) *(Lim-Ok* ∧ *L-Ok* ∧ *S-Ok* ∧ *P-Ok* ∧ *state=Test.And.Init)*
(b) *(Lim-Ok* ∧ *L-Ok* ∧ *S-Ok* ∧ *P-Ok* ∧ *state=Normal)*
(c) *(Lim-Ok* ∧ *L-Ok* ∧ *S-Ok* ∧ *P-Ok* ∧ *state=Degraded)*
(d) *(Lim-Ok* ∧¬ *L-Ok* ∧ *state = Normal)*
(e) *(Lim-Ok* ∧¬ *L-Ok* ∧ *state = Degraded)*

temos que provar que a fórmula ((a) ∨ (b) ∨ (c)) ∧ ((d) ∨ (e)) é igual a *false*. Primeiro podemos escrever ((a) ∨ (b) ∨ (c)) como

$$(f)\ \mathit{Lim\text{-}Ok} \wedge \mathit{L\text{-}Ok} \wedge \mathit{S\text{-}Ok} \wedge \mathit{P\text{-}Ok} \wedge (\mathit{state}=\mathit{Test.And.Init} \vee \mathit{state}=\mathit{Normal} \vee \mathit{state}=\mathit{Degraded}).$$

Desta forma, usando novamente as regras de distribuição da conjução em (f) ∧ ((d) ∨ (e)), chegamos a contradições do tipo $\mathit{L\text{-}Ok} \wedge \neg\ \mathit{L\text{-}Ok}$ e $((\mathit{state} = \mathit{Test.And.Init} \vee \mathit{state} = \mathit{Degraded}) \wedge \mathit{state} = \mathit{Normal})$, e assim, obtemos como resultado final o valor *false*.

Repetindo este procedimento para cada par de estados, provamos que a especificação em Z do *Boiler System* satisfaz o Lema 2, como queríamos demonstrar. □

Segurança constitui outra importante propriedade que deve ser provada sobre a especificação em Z. Podemos formalizar a noção de segurança estabelecendo que, caso alguma *condição de risco* seja satisfeita, o sistema deve assumir o estado **shutdown**. Uma *condição de risco* é a disjunção das pré-condições extraídas dos esquemas *Test.And.Init-To-Shutdown*, *Normal-To-Shutdown*, *Degraded-To-Shutdown* e *Emergency-To-Shutdown*, que modelam as transições para o estado **shutdown**.

Definição (Condição de Risco):

$$\begin{aligned}
&\mathit{Condição\text{-}Risco} \stackrel{\text{def}}{=} \\
&\neg\ \mathit{Lim\text{-}Ok} \wedge (\mathit{state}=\mathit{Test.And.Init} \vee \mathit{state}=\mathit{Normal} \vee \mathit{state}=\mathit{Degraded} \vee \mathit{state}=\mathit{Emergency}) \\
&\quad \vee \\
&\mathit{Lim\text{-}Ok} \wedge (\neg\ \mathit{L\text{-}Ok} \vee \neg\ \mathit{S\text{-}Ok}) \wedge \mathit{state}=\mathit{Test.And.Init} \\
&\quad \vee \\
&\mathit{Lim\text{-}Ok} \wedge \mathit{L\text{-}Ok} \wedge \mathit{S\text{-}Ok} \wedge \neg\ \mathit{P\text{-}Ok} \wedge \mathit{np}<3 \wedge \mathit{state}=\mathit{Test.And.Init} \\
&\quad \vee \\
&\mathit{Lim\text{-}Ok} \wedge \neg\ \mathit{L\text{-}Ok} \wedge \mathit{state}=\mathit{Emergency}
\end{aligned}$$

Assim, definimos o seguinte Teorema

Teorema 2: $\mathit{Condição\text{-}Risco} \Rightarrow \mathit{state}' = \mathit{Shutdown} \wedge \mathit{BSys\text{-}Off}$

onde $\mathit{Bsys\text{-}Off} \stackrel{\text{def}}{=} \mathit{up}' = \mathit{OFF} \wedge \mathit{valve}' = \mathit{OPEN} \wedge \mathit{ran\ pumps}' = \{\mathit{OFF}\}$

O Teorema 2 é facilmente provado em [4], pelo método tableau ([3]).

Com a prova do Teorema 2, estabelecemos um critério de segurança para a especificação em Z, tomada isoladamente.

A partir dos teoremas e lemas acima, concluimos que a especificaçãoem Z define um sistema de transição de estados que não entra em *deadlock* (Lema 1), é determinístico (Lema 2) e seguro (Teorema 2). Estas propriedades não são expressas na especificação em TLA, ou seja, construímos uma especificação segura e que estende nossa compreensão do sistema, como era nosso objetivo inicial.

Embora a especificação em Z tenha descrito alguns aspectos importantes que não estão na especificação em TLA, concorrência e comunicação ainda não foram especificados, e portanto, a utilização de outros formalismos como álgebra de processos para descrever tais aspectos pode tornar a análise de requisitos do *Boiler System* ainda mais completa. Isto é feito em [4].

5 Relacionando as Especificações

Nesta seção queremos mostrar que, quando utilizadas em conjunto, a especificação em TLA da Seção 3 e a especificação em Z da Seção 4 não são *conflitantes*, ou seja, não existe contradição lógica entre as duas descrições. Desta forma, obtemos uma especificação multiformalismo consistente e mais completa.

Primeiramente, consideramos a especificação em TLA do *Boiler System*. Dentre algumas propriedades de segurança provadas em [2] sobre esta especificação, a mais importante mostra que, enquanto a variável *up* tem o valor *true*, o nível de água do boiler permanece dentro de seu limite de segurança.

Assim, o seguinte Teorema foi provado em [2]

$$BSys \Rightarrow Safety$$

onde a noção de segurança é formalizada por

$$Safety \stackrel{\text{def}}{=} \Box(up \Rightarrow L \subseteq Safe)$$

Esta propriedade deve ser preservada pela especificação em Z, quando utilizamos as duas especificações conjuntamente. Contudo, ao tratarmos com dois formalismos diferentes como Z e TLA, não podemos estabelecer propriedades que relacionam as duas especificações diretamente. Sabemos apenas que os dois modelos representam aspectos diferentes do sistema: a especificação em TLA modela as propriedades de segurança do *Boiler System* (de acordo com o modelo de falhas descutido anteriormente), enquanto que a especificação em Z modela o sistema como uma máquina de transição de estados.

Assim, para podermos relacionar as duas especificações, traduzimos o modelo em Z das transições de estados do *Boiler System* para TLA, preservando o significado da especificação em Z, e possibilitando checagem de consistência entre os dois diferentes modelos do sistema. A definição genérica desta tradução é definida de forma rigorosa em [4].

Desta forma, podemos representar as transições de estados modelados em Z como a seguinte fórmula em TLA:

$$BSys_Z \stackrel{\text{def}}{=} Init_Z \wedge \Box Transitions \wedge \Box Inv$$

onde, $Init_Z$ representa as condições iniciais do sistema (em Z representadas pelo esquema *Self.Test*), *Transitions* representa todas as transições de estado da Figura 2 e Inv representa as propriedades invariantes que o sistema deve satisfazer (extraídas do esquema *BSys*).

$$Init_Z \stackrel{\text{def}}{=} L'_c = 0 \wedge S'_c = 0 \wedge state' = Self.Test$$

$$Transitions \stackrel{\text{def}}{=} Trans_1 \vee Trans_2 \vee ... \vee Trans_{13}$$

onde $Trans_1, Trans_2, ..., Trans_{13}$ representa todas as transições da Figura 2. Por exemplo, $Trans1 = Self.Test\text{-}To\text{-}Test.And.Init$, definida por

$$\begin{aligned} Trans_1 \stackrel{\text{def}}{=}\ & \neg valve \wedge \neg up \wedge state = Self.Test \\ & \wedge\ valve' \wedge up' \wedge state' = Test.And.Init \\ & \wedge \bigwedge_{i \in D_p} \neg pumps \\ & \wedge \bigwedge_{i \in D_p} pumps' \end{aligned}$$

onde, D_p equivale ao domínio do conjunto de pares *pumps*.

$$
\begin{aligned}
Inv \stackrel{def}{=} & \sharp(pumps) = np \\
& \wedge S \leq S_p \\
& \wedge P \leq P_p \\
& \wedge L_c = L + (P - S) \\
& \wedge S_c \leq S_p
\end{aligned}
$$

Assim, após a tradução do modelo Z em TLA, precisamos mostrar que a noção de segurança *Safety* descrita no início desta seção é preservada por $BSys_Z$, de forma que $BSys_Z \Rightarrow Safety$. Desta forma, podemos afirmar que a especificação em Z garante propriedades de segurança da especificação em TLA, e portanto, quando utilizadas em conjunto, as especificações não são contraditórias. Esta prova é desenvolvida em [4].

O leitor atento poderia questionar se não seria necessário mostrar que a especificação em TLA respeita o Teorema 2 da seção 4. Esta prova, no entanto, não é necessária uma vez que a especificação em TLA *não* define transições de estado e, portanto, não pode contradizer o Teorema 2. Desta forma, usamos o mesmo argumento para explicar que a especificação em TLA também não pode contradizer o Teorema 1.

6 Considerações Finais

O objetivo básico deste artigo é mostrar que a utilização de múltiplos formalismos permite especificar sistemas de segurança crítica de forma mais completa e confiável.

Para isso, usamos como exemplo um sistema de caldeiras (*Boiler System*), descrito em dois modelos distintos pelos formalismos TLA e Z. Mostramos que a especificação em TLA, tomada isoladamente, deixa de considerar aspectos importantes do sistema como falhas isoladas de dispositivos, comunicação e concorrência. Com a especificação em Z, aumentamos o conhecimento sobre o sistema, tornando a especificação do *Boiler System* mais completa e, com as provas de consistência entre as duas especificações, mais confiável, embora aspectos de comunicação e concorrência não tenham ainda sido tratados.

Portanto, comprovamos, teoricamente (visto que não chegamos a uma implementação do *Boiler System*), a viabilidade e adequação da utilização de múltiplos formalismos para desenvolvimento de sistemas de segurança crítica.

Alguns aspectos como o custo de desenvolvimento formal e o uso de ferramentas de apoio não são discutidos aqui, embora reconhecemos a importância de tais considerações.

Nosso próximo passo é especificar aspectos de concorrência e comunicação do *Boiler System*, produzir a integração destes aspectos com os aspectos abordados neste artigo e automatizar algumas provas com a codificação das lógicas em provadores de teorema.

Agradecimentos

Agradecemos Augusto C. A. Sampaio, Ismar N. Kaufman e Ana T. C. Martins por vários comentários importantes sobre o trabalho descrito neste artigo.

Referências

[1] L. M. Barroca and J. A. McDermid. Formal methods: Use and relevance for the development of safety-critical systems. *The Computer Journal vol 35(6)*, pages 579–599, 1992.

[2] Glenn Bruns and Stuart Anderson. Using data consistency assumptions to show system safety, 1993. Laboratory for Foundations of Computer Science, Edinburgh.

[3] Marcos Mota do Carmo Costa. *Introdução à Lógica Modal Aplicada à Computação.* Informática UFRGS, Departamento de Informática UFPE, 1992.

[4] Simone C. dos Santos. Especificação formal no desenvolvimento de sistemas de segurança crítica (título preliminar). Dissertação de mestrado em andamento, Departamento de Informática, UFPE, 1995.

[5] Narain Gehani and Andrew McGettrick. *Software Specification Techniques.* International Computer Science Series, N. J., 1986.

[6] Farnam Jahanian and Aloysius Ka-Lau Mok. Safety analysis of timing properties in real-time. *IEEE Transactions on Software Engineering*, pages 890–901, 1986.

[7] M. B. Josephs. A state-based approach to communicating processes. *Distributed Computing*, pages 9–18, 1988.

[8] Ismar N. Kaufman. A aplicação de especificações formais no projeto lógico de software, 1992. Dissertação de Mestrado, Departamento de Informática, UFPE.

[9] Leslie Lamport. The temporal logic of actions. Technical report, 1991. Digital Systems Research Center.

[10] Nancy G. Leveson. Software safety: what, why and how. *Computing Surveys vol 18(2)*, pages 125–163, 1986.

[11] Nancy G. Leveson and Janice L. Stolzy. Safety analysis using petri nets. *IEEE Transactions on Software Engineering*, pages 386–397, 1987.

[12] A. Saeed R. de Lemos and T. Anderson. A train as a case study for the requirements analysis of safety-critical systems. *Computer Journal vol 35(1)*, pages 30–39, 1992.

[13] Institute For Risk Research. Specification for a software program for a boiler water content monitor and control system, 1992. Waterloo, Canadá.

[14] J. M. Spivey. *The Z notation: A reference manual.* Prentice-Hall, Englewood Cliffs NJ, 1989.

[15] P. A. Lee T. Anderson and editors. *Fault Tolerance: Principles and Practice.* Prentice Hall, 1981.

[16] J. B. Wordsworth. *Software Development with Z.* Addison-Wesley, England, 1992.