Revisão de Crenças Temporais

Paulo de Tarso Guerra Oliveira

Tese apresentada
ao
Instituto de Matemática e Estatística
da
Universidade de São Paulo
para
obtenção do título
de
Doutor em Ciências

Programa: Ciência da Computação

Orientador: Profa. Dra. Renata Wassermann

Durante o desenvolvimento deste trabalho o autor recebeu auxílio financeiro da FAPESP, processo nº 2010/15392-3, Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP).

São Paulo, janeiro de 2016

Revisão de Crenças Temporais

Esta é a versão original da tese elaborada pelo
candidato Paulo de Tarso Guerra Oliveira, tal como
submetida à Comissão Julgadora.

Resumo

GUERRA, P. T. **Revisão de Crenças Temporais**. 2016. f. Tese (Doutorado) - Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2016.

Lógicas temporais são a base da área de verificação formal de sistemas. Em verificação formal, um sistema é descrito em uma linguagem formal e verificado frente um conjunto de propriedades desejadas, usualmente descritas por fórmulas de uma lógica modal. Apesar de técnicas de verificação formal poderem lidar com modelos complexos de sistema, ferramentas de verificação usualmente não auxiliam no processo de reparar uma especificação inconsistente. O objetivo intuitivo é remover tal inconsistência com mudanças mínimas na especificação original.

Revisão de crenças trata do problema de como um agente racional deve adaptar suas crenças de modo a incorporar novas informações. Quando crenças e nova informação são inconsistentes, essa adaptação envolve desistência de crenças de modo que a consistência seja restaurada. A Teoria AGM de revisão de crenças propõe princípios básicos de racionalidade para esse processo, para que nessa adaptação de crenças o agente, por exemplo, não desista de crenças desnecessariamente.

Trabalhos recentes em verificação formal desenvolvem abordagens que enriquecem métodos de verificação com princípios de mudanças de crenças, de modo a, por exemplo, modificar modelos de sistemas para que se tornem consistentes com uma dada propriedade temporal. Todavia, uma aplicação completa da teoria de revisão de crenças demanda a reformulação de diversos resultados clássicos para o formalismo de lógicas temporais. O problema ocorre porque lógicas temporais não satisfazem suposições clássicas, como o fato de serem lógicas compactas.

Neste trabalho investigamos o uso de revisão de crenças sobre lógicas temporais, como objetivo criar fundamentações teóricas para a completa utilização desta teoria no reparo de especificações inconsistentes. Mostramos que o problema do reparo de especificação é dependente da perspectiva adotada pelo projetista sobre o significado desta especificação. Mostramos que fatores relativos a especificação de sistemas demandam duas abordagens distintas de revisão da especificação: uma abordagem baseada em conjuntos de fórmulas e outra baseada em modelos.

A revisão de conjuntos de fórmulas temporais é uma abordagem semelhante as abordagens clássicas de revisão de crenças. Contudo, mostramos que no caso geral tal abordagem é incomputável para lógicas temporais. Propomos então restrições ao problema que possibilitam a formulação de operadores de revisão de crenças e mostramos que essas restrições ainda são aplicáveis a diversos problemas práticos de verificação formal.

A abordagem de revisão de modelos é a adotada em diversos trabalhos recentes. Mostramos que operações desse tipo podem ser caracterizadas por postulados de racionalidade semelhantes aos propostos na Teoria AGM. Investigamos também revisão de modelos no contexto de especificações parciais, estabelecendo a caracterização de princípios de racionalidade para este cenário.

Palavras-chave: revisão de crenças, lógica temporal, verificação de modelos.

Abstract

GUERRA, P. T. **Revision of Temporal Beliefs**. 2016. f. Tese (Doutorado) - Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2010.

Temporal logics are the basis of the area of formal system verification. In formal verification, a system is described in a formal language and then checked against a set of desired properties, usually described in some temporal logic formalism. Although this technique can handle complex verifications, formal verification tools usually do not give any information on how to repair inconsistent system models. Once an inconsistency is detected, the intuitive goal is to remove this inconsistency through minimal changes in the original specification.

Belief revision addresses the problem of how idealized agents should change their beliefs when receiving new information. When this information is inconsistent with the agent's beliefs, this change may involve withdrawal of some beliefs in order to restore consistency. The AGM theory of belief revision propose a minimality rationality principles that any rational agent should satisfies.

Recent works in formal verification build approaches that improve formal verification methods with some principles of AGM or other related change theories. These approaches modify a model of a system, in order to make it consistent with some temporal formula. However, the full application of AGM theory demands a reformulation of several classical results for the temporal formalism. The AGM theory of belief revision has been successfully applied to families of logics satisfying certain assumptions, as being compact. For non-compact logics, which includes most of the temporal logics, there are still no general results that can be used.

In this work we investigate the application of the AGM paradigm to temporal logics. The goal is to establish theoretical foundations for the full use of this theory in the repair of inconsistent specifications. We show that the problem of specification repair is dependent of the perspective adopted by the designer with respect to the specification. We show that the repair problem demands two distinct approaches: a revision approach based on sets of formulas and other based on model.

The revision of sets of temporal formulas is an approach similar to the classic approaches in belief revision. We show, however, that such approach is incomputable to temporal logics in the general case. We then propose restrictions on the problem such that we could correctly formulate belief revision operators to temporal logic. We show although the restriction, our approach could still be applied to several problems in formal verification.

The model-based approach is related to that used by most of the recent works in automated model repair. We show that repair operations can be characterized by AGM-style rationality postulates. We also explore the model revision approach for partial specifications, giving also the characterizations properties in this context.

Keywords: belief revision, temporal logic, model checking.

Sumário

Lista de Abreviaturas

ALL	Autômato Linearmente Limitado
BNF	Formalismo de Backus-Naur (*Backus-Naur Form*)
CTL	Lógica de Árvore Computacional (*Computation Tree Logic*)
LTL	Lógica de Tempo Linear (*Linear-time Temporal Logic*)
LTS	Sistema de Transição Rotulado
KMTS	Sistema de Transição Modal de Kripke (*Kripke Modal Transition System*)
PDL	Lógica Dinâmica Proposicional (*Propositional Dynamic Logic*)

Lista de Figuras

Capítulo 1

Introdução

Lógicas temporais são um tipo de lógica modal onde podemos representar e raciocinar sobre o tempo. Duas lógicas temporais se destacam por suas aplicações. A *lógica de tempo linear* (*linear-time temporal logic*, ou LTL) [Pnu77, LP85] permite representar afirmações sobre eventos, onde o futuro é interpretado como uma sequência linear única destes eventos. A *lógica de árvore computacional* (*computation tree logic*, ou CTL) [CE82], modela o futuro como ramificações de uma árvore e por essa característica, é perfeita para raciocinar sobre possíveis caminhos de execução de um sistema. Lógicas como LTL e CTL são as bases da área de verificação formal de sistemas, sendo o formalismo utilizado por diversos métodos para assegurar a correção de modelos de sistemas.

Uma das técnicas mais eficientes de verificação formal é chamada *verificação de modelos* [CES86, CGP99]. Esta técnica permite ao projetista de sistemas verificar quando uma especificação formal de um sistema satisfaz (ou não) um conjunto de propriedades (fórmulas temporais). Contudo, a maioria das ferramentas de verificação, quando encontram um erro, apenas informam como este pode ser encontrado (caminho contraexemplo). Contudo, essa informação pode não ser tão informativa em grandes projetos. Quanto mais complexo o sistema, mais difícil é para o projetista corrigi-lo. Em geral, ferramentas de verificação de modelos não possuem mecanismos capazes de auxiliar o usuário na tarefa de correção, o que não as credencia como ferramentas completas para o reparo de sistemas.

Revisão de crenças [AGM85] é um tópico bastante estudado em Inteligência Artificial. Esta técnica se refere ao modo como um agente deve se comportar ao agregar novas informações a suas crenças, mesmo que elas sejam conflitantes com o que o agente acredita. Se virmos o modelo verificado como nossa crença e a propriedade inconsistente como a informação a ser agregada, essa técnica se torna uma poderosa ferramenta para o reparo automático de inconsistências.

Entretanto, nem todos os resultados conhecidos de revisão podem ser aplicados a crenças temporais. Por exemplo, a garantia da existência de um operador de revisão construído segundo o paradigma AGM é assegurada para lógicas monotônicas e compactas [HW02], mas a lógica CTL não possui esta última propriedade[1].

O objetivo principal do nosso trabalho é explorar revisão de crenças em sobre lógicas do tipo temporal, em especial para as lógicas LTL e CTL. Entender os efeitos da técnica nessa abordagem contribui para a expansão seu uso além do escopo tradicional, permitindo que estas futuras aplicações sejam fundamentadas em princípios clássicos de revisão de crenças.

1.1 Trabalhos Relacionados

O trabalho [BEGL99] foi o primeiro nessa linha. Buccafurri *et al.* desenvolveram um arcabouço formal integrando verificação de modelos e revisão abdutiva de teorias para realizar o diagnóstico e reparo de erros em programas concorrentes. O raciocínio abdutivo é utilizado para encontrar modificações no sistema a fim de que este satisfaça todas as propriedades de uma especificação formal.

[1] Uma lógica é compacta se para todo α inferido do conjunto de fórmulas A, existe um conjunto finito $B \subseteq A$ tal que α é inferido de B.

Nessa abordagem, um sistema é modelado segundo uma estrutura de Kripke[2] e as modificações correspondem a uma sequência de adições ou remoções de transições de estados que torna o modelo consistente com a especificação.

Apesar da abordagem de [BEGL99] ser bem sucedida para seus propósitos, o conceito de modificação adotado é um tanto quanto restrito. Zhang e Ding [ZD08] propõem um arcabouço para atualização de modelos que supre as deficiências apresentadas pelo método de Buccafurri *et al.*.

A abordagem de [ZD08] é baseada na integração de verificação de modelos e *atualização de crenças* [HR99, KM91]. Os autores especificaram o princípio da mudança mínima para atualização de modelos e então definiram o conceito de atualização admissível. Os autores também especificaram um procedimento para a atualização de modelos e analisaram suas propriedades semânticas e computacionais.

Em [GW10] argumentamos que uma abordagem baseada em atualização de crenças não é apropriada para todos os casos. Propomos o uso de *revisão de crenças* [AGM85] para mudanças em modelos quando o reparo de especificações ocorrer em um contexto estático. Apesar da similaridade entre atualização de crenças e revisão de crenças, o uso da abordagem incorreta pode acarretar em significativa perda de informação.

Contudo, nem todos os resultados conhecidos de revisão de crenças se aplicam a nossa abordagem. A verificação de modelos baseia-se em lógicas temporais, como as lógicas LTL e CTL, mas a maioria dos resultados obtidos assumem propriedades que essas lógicas não possuem.

Souza e Wassermann [SW07] abordam o uso prático de revisão de crenças para o reparo de modelos inconsistentes. Os autores criaram uma ferramenta capaz de gerar sugestões de reparo para modelos descrito segundo a linguagem SMV. Acreditamos, no entanto, que a técnica poderia explorar melhor os conceitos de revisão, sendo possível desenvolver uma abordagem que possua fundamentos sólidos na teoria clássica de revisão de crenças.

Em [Gue10] descrevemos o conceito de revisão de modelos CTL: uma abordagem baseada em revisão de crenças para o reparo de modelos inconsistentes em um contexto estático. Nós relacionamos nossa proposta a alguns trabalhos clássicos em revisão de crenças e discutimos questões de como implementá-la.

Neste projeto propomos ir mais a fundo nesta análise, investigando as implicações da adaptação da técnica de revisão a esse formalismo lógico. Propomos a formulação de um arcabouço para revisão de crenças compostas por fórmulas temporais, bem como investigar a viabilidade de aplicá-lo a lógicas com características semelhantes.

Contribuições

O objetivo principal do nosso trabalho é formalizar a aplicação de revisão de crenças em lógicas temporais. Entender os efeitos da técnica nessa abordagem contribui para sua expansão para além do escopo tradicional, permitindo que futuras aplicações para reparo automático possam ser fundamentadas em princípios clássicos de revisão de crenças.

As principais contribuições deste trabalho são:

1. Separação do problema em duas abordagens distintas: revisão de fórmulas temporais e revisão de modelos temporais;

2. Mostrar que as duas abordagens não são inter-definíveis e que seu uso incorreto pode levar a perca de informação;

3. Demonstração da incomputabilidade do problema da revisão de conjuntos de fórmulas temporais;

4. Definição de fragmentos computáveis para o problema da revisão de conjuntos de fórmulas temporais, por meio de restrições na entrada do problema;

[2] Detalhes sobre estruturas de Kripke serão abordados no Capítulo 2.

5. Revisão de modelos de especificações parciais por meio do reparo de estruturas modais de Kripke;

1.2 Organização

Nos dois próximos capítulos apresentamos conceitos preliminares necessários ao desenvolvimento do trabalho. No Capítulo 2 apresentamos o formalismo de lógicas temporais, com ênfase nas lógicas LTL e CTL, sua linguagem e semântica.

No Capítulo 3 apresentamos a teoria AGM de revisão de crenças. Apresentamos seu sistema de crenças, postulados de racionalidade e construções de operadores de mudança de crenças. Apresentamos também diferentes formas de representar crenças, como bases de crenças e mundos possíveis.

No Capítulo 4 abordamos o problema da revisão de crenças voltada ao reparo de especificações de sistemas. Definimos as duas abordagens possíveis ao problema e mostramos o porquê destas abordagens necessitarem de um desenvolvimento independente.

No Capítulo 5 mostramos que o problema da revisão sobre bases de formulas temporais é incomputável. Definimos restrições ao problema aos quais podemos corretamente definir revisão e mostramos que tais revisões são ainda úteis para o problema do reparo de especificações.

No Capítulo 6 mostramos a segunda abordagem proposta, a revisão de modelos, pode ser caracterizada com base em princípios clássicos de revisão de crenças. Mostramos que operadores de revisão de modelos podem ser caracterizados por um conjunto de postulados de racionalidade nos termos de revisão de modelos.

No Capítulo 7 mostramos que para especificações parciais, onde uma especificação aceita diversos possíveis modelos concretos, a revisão de crenças produz melhores resultados se comparado a atualização de modelos de [ZD08], quando aplicada um ambiente estático. Mostramos como realizar revisão de crenças por meio de estruturas modais de Kripke (KMTS). Descrevemos propriedades desta abordagem e propomos algoritmos para o reparo.

Por fim, no Capítulo 8 apresentamos nossas conclusões, discussões sobre trabalhos relacionados e apontamos possibilidades de trabalhos futuros.

1.3 Notação

Denotamos por $\mathcal{L}_{Prop}$ para denotar a linguagem proposicional fechada sobre os conectivos lógicos $\neg, \wedge, \vee, \rightarrow$, e com a constante $\bot$ representando o valor *falso*. Denotamos por $\mathcal{L}_{LTL}, \mathcal{L}_{CTL}$ e $\mathcal{L}_{\mu}$, as linguagens das lógicas LTL, CTL e μ-calculus, respectivamente, segundo as definições dadas no Capítulo 2. Utilizamos letras gregas minúsculas $(\alpha, \beta, \phi, \varphi, \psi)$ para denotar fórmulas de uma linguagem $\mathcal{L}$. Conjuntos de fórmulas são denotados por letras maiúsculas $(A, B, K, X, ...)$. Átomos proposicionais em uma linguagem $\mathcal{L}$ são denotadas por letras minúsculas $(p, q, a, b, ...)$. A consequência lógica é representado por um operador Cn, onde $Cn(A) = B$ e B contém ao menos as consequências clássicas da lógica proposicional. Utilizamos letras maiúsculas caligráficas $(\mathcal{M}, \mathcal{N}, \mathcal{I}, \mathcal{J}, ...)$ para denotar modelos semânticos, como sistemas de transições rotulados. Letras minúsculas, possivelmente com subscritos, também são usadas para denotar estados nestes modelos $(s, r, s_0, s_1, ...)$. Os demais símbolos são apresentados ao longo do texto.

Capítulo 2

Lógicas Temporais

Lógicas temporais [Pnu77, LP85, CE82] são um tipo de lógica modal onde podemos representar e raciocinar sobre o tempo. Lógicas temporais compõe a base da área de verificação formal de sistemas, sendo usada para descrever as propriedades que são esperadas de um sistema durante seu tempo de sua execução.

2.1 Lógica de Tempo Linear

A Lógica de Tempo Linear (LTL) [Pnu77, LP85] é uma lógica temporal utilizada sobre uma interpretação de futuro linear, onde o tempo é entendido como uma sequência única de eventos. Essa interpretação contrasta com interpretação de futuro ramificado, apresentada na seção 2.2.

A lógica LTL é lógica modal supraclássica, contendo todos os axiomas clássicos também são válidos neste formalismo. Em LTL, as modalidades são utilizadas para se estabelecer afirmações sobre estados (eventos) na linha do tempo. Os principais conectivos são X (no próximo estado), F (em um estado futuro), G (em todos os estados), U (verdadeiro até que) e R ("liberado por").

A sintaxe formal de LTL é dada pela seguinte BNF

$$\phi ::= \bot \mid p \mid \neg\phi \mid \phi \vee \phi \mid \mathrm{X}\phi \mid \phi \mathrm{U} \phi \mid \phi \mathrm{R} \phi$$

onde $\top \equiv \neg\bot$, $\phi_1 \wedge \phi_2 \equiv \neg(\neg\phi_1 \vee \neg\phi_2)$, $\phi_1 \rightarrow \phi_2 \equiv \neg\phi_1 \vee \phi_2$, $\mathrm{F}\phi \equiv \top \mathrm{U} \phi$ e $\mathrm{G}\phi \equiv \bot \mathrm{R} \phi$.

A semântica de LTL é dada por meio de sistemas de transição rotulado (*labeled transition system*, ou LTS) que finitamente representa uma sequência infinita de estados. Um modelo LTL é um LTS com as seguintes características

Definição 1. Um *modelo LTL* é um sistema de transição rotulado $\mathcal{M} = \langle AP, S, s_0, R, L \rangle$ onde

1. AP é um conjunto contável de átomos proposicionais,
2. S é um conjunto finito de estados,
3. s_0 é o estado inicial
4. $R \subseteq S \times S$ é uma relação de transição sobre S serial e unívoca (funcional)[1],
5. $L : AP \rightarrow \mathcal{P}(S)$ é uma função de rotulação de estados que indica em quais estados uma proposição é verdadeira.

A Figura 2.1 ilustra um diagrama de estados de um modelo LTL formalmente definido como $M = \langle \{p, q\}, \{s_0, s_1\}, s_0, \{(s_0, s_1), (s_1, s_1)\}, L \rangle$ onde $L(p) = \{s_0, s_1\}$ e $L(q) = \{s_1\}$.

Nossa definição de modelo LTL a princípio é mais restritiva que a definição de semântica LTL por meio de estruturas de Kripke (Definição 5), mais comumente encontrada na literatura. Contudo

[1] R é serial se e somente se para todo $a \in S$, $(a, b) \in R$. R é unívoca se e somente separa todo $\{(a, b), (a, c)\} \subseteq R$, $b = c$.

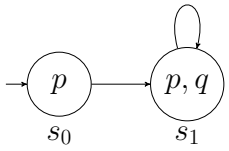


Figura 2.1: *Exemplo de modelo LTL.*

a propriedade de recorte (*snipping lemma*) [GHR94] de LTL mostra que tal modelo existe para qualquer fórmula LTL satisfazível. Nossa escolha por esta forma de definir modelos LTL aproxima-os da interpretação de sequência única de eventos, simplificando em diversos pontos as provas apresentadas no Capítulo 5.

Um *caminho de computação* em um modelo LTL é uma sequência de estados deste modelo, representando os eventos na linha de tempo, que seguem a seguinte definição

Definição 2. Seja $\mathcal{M} = \langle AP, S, s_0, R, L\rangle$ um modelo LTL. Um caminho de computação $\pi = s_0 \to s_1 \to ...$ em $\mathcal{M}$ é uma sequência de estados $\mathcal{M}$ tal que, para todo $i \geq 0$, $(s_i, s_{i+1}) \in R$.

A relação de satisfação $\models$ entre estados de um modelo e fórmulas LTL é definida como segue

Definição 3. Seja $\mathcal{M} = \langle AP, S, s_0, R, L\rangle$ um modelo LTL e $s \in S$ um estado de $\mathcal{M}$. A relação de satisfação $\mathcal{M}, s \models \phi$ é definida por indução estrutural em ϕ como segue

1. $\mathcal{M}, s \not\models \bot$
2. $\mathcal{M}, s \models p$ se e somente se $s \in L(p)$
3. $\mathcal{M}, s \models \neg\phi$ se e somente se $\mathcal{M}, s \not\models \phi$
4. $\mathcal{M}, s \models \phi_1 \vee \phi_2$ se e somente se $\mathcal{M}, s \models \phi_1$ ou $\mathcal{M}, s \models \phi_2$
5. $\mathcal{M}, s \models \mathrm{X}\phi$ se e somente se $(s, s') \in R$ e $\mathcal{M}, s' \models \phi$
6. $\mathcal{M}, s \models \phi_1 \mathrm{U} \phi_2$ se e somente se no caminho $\pi = s_0 \to s_1 \to ...$, onde $s = s_0$, vale que para algum s_n, $M, s_n \models \phi_2$ e para todos $i < n$, $M, s_i \models \phi_1$.
7. $\mathcal{M}, s \models \phi_1 \mathrm{R} \phi_2$ se e somente se no caminho $\pi = s_0 \to s_1 \to ...$, onde $s = s_0$, vale que (a) $M, s_n \models \phi_1$ para algum s_n e $M, s_i \models \phi_2$ para todo $i \leq n$, ou (b) $M, s_n \models \phi_2$ para todo $i \geq 0$.

Se $M, s_0 \models \phi$, dizemos que $\mathcal{M}$ satisfaz ϕ, o qual denotamos por $\mathcal{M} \models \phi$. O modelo $\mathcal{M}$ da Figura 2.1, por exemplo, satisfaz as fórmulas $p \wedge \mathrm{X}q$, $p\mathrm{U}q$ e $\mathrm{FG}q$, mas não $q \wedge \mathrm{X}p$ ou $\mathrm{G}q$.

2.2 Lógica de Árvore Computacional

Diferente de LTL, a Lógica de Árvore Computacional (*computation tree logic*, ou CTL) [CE82] é uma lógica temporal que modela o futuro como ramificações de uma árvore. Essa característica a torna especialmente apropriada para o uso em verificação formal, por poder quantificar sobre os diversos caminhos de execução que um sistema pode tomar.

Definição 4 (Sintaxe)**.** Uma fórmula CTL tem a seguinte sintaxe:

$$\varphi \quad ::= \quad \top \mid p \mid (\neg\varphi) \mid (\varphi \wedge \varphi) \mid EX\varphi \mid EG\varphi \mid E[\varphi U \varphi]$$

onde p é uma fórmula atômica, $\neg$, $\wedge$ os conectivos lógicos clássicos e os demais elementos os operadores temporais.

Usamos $\bot$ para denotar $\neg\top$, e outros operadores temporais podem ser derivados de EX, EG e EU: $\text{AX}\phi = \neg\text{EX}\neg\phi$; $\text{AG}\phi = \neg\text{EF}\neg\phi$; $\text{AF}\phi = \neg\text{EG}\neg\phi$; $\text{EF}\phi = \text{E}[\top\text{U}\phi]$; $\text{A}[\phi\text{U}\beta] = \neg\text{E}[\neg\beta\text{U}\neg\phi \wedge \neg\beta] \wedge \neg\text{EG}\neg\beta$.

Cada operador temporal consiste de um quantificador de caminho (E, "existe um caminho", ou A, "para todos os caminhos") seguido por um quantificador de estado (X, "próximo estado", U, "até que", G, "todos os estados" ou F, "algum estado futuro").

A semântica para a lógica LTL é definida sobre modelos denominados *estrutura de Kripke* [CH12]. Uma estrutura de Kripke é um tipo de sistema de transição rotulado, semelhante ao apresentado na Definição 1, exceto por não exigir que a relação de transição seja funcional. Formalmente, uma estrutura de Kripke é definida como

Definição 5. Seja AP um conjunto finito de proposições, um *modelo de Kripke* é uma tripla $M = (S, R, L)$, onde

1. S é um conjunto de estados
2. $R \in S \times S$ é uma relação serial entre estados
3. $L : AP \to \mathcal{P}(S)$ é uma função de rotulação de estados que indica em quais estados uma proposição é verdadeira.

Deste ponto em diante, sempre que nos referirmos à um modelo CTL estamos na verdade nos referindo a um modelo de Kripke conforme a Definição 5.

Definição 6 (Semântica)**.** Seja $M = (S, R, L)$ um modelo CTL, $s \in S$ um estado de M e ϕ uma fórmula CTL. Nós definimos $(M, s) \vDash \phi$ indutivamente como

1. $(M, s) \vDash \top$.
2. $(M, s) \vDash p$ se e somente se $s \in L(p)$.
3. $(M, s) \vDash \neg\phi$ se e somente se $(M, s) \nvDash \phi$.
4. $(M, s) \vDash \phi_1 \wedge \phi_2$ se e somente se $(M, s) \vDash \phi_1$ e $(M, s) \vDash \phi_2$.
5. $(M, s) \vDash EX\phi$ se e somente se $\exists s' \in S$ tal que $(s, s') \in R$ e $(M, s') \vDash \phi$.
6. $(M, s) \vDash EG\phi$ se e somente se existe um caminho $\pi = [s_0, s_1, \dots]$ em M tal que $s_0 = s$ e $(M, s_i) \vDash \phi$ para todo $i \geq 0$.
7. $(M, s) \vDash E[\phi_1 U \phi_2]$ se e somente se existe um caminho $\pi = [s_0, s_1, \dots]$ em M tal que $s_0 = s$, $\exists i \geq 0$, $(M, s_i) \vDash \phi_2$ e $\forall j < i$, $(M, s_j) \vDash \phi_1$.

Dado um modelo CTL $M = (S, R, L)$ e seu conjunto de estados iniciais $Init(S) \subseteq S$, dizemos que $M \vDash \phi$, se e somente se $(M, s) \vDash \phi$ para algum $s \in Init(S)$.

2.3 Outras Lógicas Temporais

O universo de lógicas temporais vai bem além das lógicas LTL e CTL. A lógica CTL*, por exemplo, é uma lógica temporal que combina a expressividade de LTL e CTL. A lógica CTL* é então definida do mesmo modo que a lógica CTL, exceto por não possuir a limitação que cada operador temporal seja seguido de um quantificador de caminho. Assim, em CTL* as seguintes formulações são possíveis.

$$\text{G}(\text{EF}p)$$
$$\text{E}(p\text{UF}q)$$
$$\text{A}((\text{FG}p) \to \text{EF}q)$$

Neste contexto, as lógicas LTL e CTL são denominados fragmentos da lógica CTL*. A lógica CTL* por sua vez é fragmento de uma lógica modal mais expressiva, denominada *μ-calculus*.

2.3.1 Lógica μ-calculus

A lógica *μ-calculus* é uma lógica modal enriquecida com operadores de maior e menor ponto-fixo. Sua sintaxe é dada por

Definição 7. Seja AP um conjunto de átomos proposicionais, V um conjunto de símbolos de variáveis e A um conjunto de ações, uma fórmula na linguagem $\mathcal{L}_\mu$ é dada pela seguinte gramática

$$\begin{aligned}\varphi ::= \top \mid \bot \mid p \mid \neg p \mid X \mid (\varphi \wedge \varphi) \mid (\varphi \vee \varphi) \mid \\ \langle a \rangle \varphi \mid [a]\varphi \mid \mu X.\varphi \mid \nu X.\varphi\end{aligned}$$

onde $p \in AP$, $X \in V$ e $a \in A$.

A semântica de *μ-calculus* é dada também sobre estruturas de Kripke. Sua relação de satisfação é dada como segue.

Definição 8. Seja $\mathcal{M} = \langle AP, S, s_0, R, L\rangle$ um modelo de Kripke e $\phi \in \mathcal{L}_\mu$, dizemos que $\mathcal{M}$ satisfaz ϕ, denotado por $\mathcal{M} \models \phi$, se e somente se $s_0 \in [\![\phi]\!]^{\mathcal{M}}_{\xi}$ onde

- $[\![\bot]\!]^{\mathcal{M}}_{\xi} = \emptyset$
- $[\![\top]\!]^{\mathcal{M}}_{\xi} = S$
- $[\![X]\!]^{\mathcal{M}}_{\xi} = \xi(X)$
- $[\![p]\!]^{\mathcal{M}}_{\xi} = \{s \in S \mid s \in L(\phi)\}$
- $[\![\neg p]\!]^{\mathcal{M}}_{\xi} = S \setminus [\![p]\!]^{\mathcal{M}}_{\xi}$
- $[\![\phi_1 \wedge \phi_2]\!]^{\mathcal{M}}_{\xi} = [\![\phi_1]\!]^{\mathcal{M}}_{\xi} \cap [\![\phi_2]\!]^{\mathcal{M}}_{\xi}$
- $[\![\phi_1 \vee \phi_2]\!]^{\mathcal{M}}_{\xi} = [\![\phi_1]\!]^{\mathcal{M}}_{\xi} \cup [\![\phi_2]\!]^{\mathcal{M}}_{\xi}$
- $[\![\langle a\rangle \phi_1]\!]^{\mathcal{M}}_{\xi} = \{s \in S \mid \text{ existe } (s,a,r) \in R \text{ e } r \in [\![\phi_1]\!]^{\mathcal{M}}_{\xi}\}$
- $[\![[a] \phi_1]\!]^{\mathcal{M}}_{\xi} = \{s \in S \mid \text{ para todo } (s,a,r) \in R \text{ temos } r \in [\![\phi_1]\!]^{\mathcal{M}}_{\xi}\}$
- $[\![\mu X.\varphi]\!]^{\mathcal{M}}_{\xi} = \bigcap\{S' \subseteq S \mid [\![\varphi]\!]^{\mathcal{M}}_{\xi[X \leftarrow S']} \subseteq S'\}$
- $[\![\nu X.\varphi]\!]^{\mathcal{M}}_{\xi} = \bigcup\{S' \subseteq S \mid S' \subseteq [\![\varphi]\!]^{\mathcal{M}}_{\xi[X \leftarrow S']}\}$

onde ξ é um mapeamento de variáveis em estados de S e $\xi[X \leftarrow S']$ é a operação de atualização de X em ξ com o valor S'.

Os operadores CTL podem, por exemplo, serem definidos em μ-calculus em temos de menor e maior ponto fixo. Por exemplo,

$$\begin{aligned}EF\phi &\equiv \mu Z.\phi \vee EXZ \\ AF\phi &\equiv \mu Z.\phi \vee AXZ \\ EG\phi &\equiv \upsilon Z.\phi \wedge EXZ \\ AG\phi &\equiv \upsilon Z.\phi \wedge AXZ \\ E[\phi U \phi] &\equiv \mu Z.\phi \vee (\phi \wedge EXZ) \text{ e} \\ A[\phi U \phi] &\equiv \mu Z.\phi \vee (\phi \wedge AXZ).\end{aligned}$$

Neste trabalho, nos restringimos a análise da revisão de crenças aplicada às lógicas LTL e CTL. Contudo, podemos verificar que os resultados apresentados no capítulo 5 podem ser estendidos outras lógicas temporais, como CTL* e μ-calculus.

Capítulo 3

Revisão de Crenças

Revisão de crenças trata do problema de adaptar a crença de um agente a fim de incorporar uma nova informação, possivelmente inconsistente como o que o agente costumava acreditar. Em [AGM85], Alchourrón, Gärdenfors e Makinson propuseram um conjunto de postulado os quais métodos de revisão de crenças devem satisfazer, bem como construções de funções de revisão que obedecem estes postulados. Este trabalho se tornou conhecido como o *Paradigma AGM*.

3.1 Sistemas de crenças

No paradigma AGM, crenças são representadas por *conjuntos de crença*: um conjunto de formulas K tal que $Cn(K) = K$, onde Cn é um operador de consequência supraclássico. Duas outras representações de crenças são importantes para nosso trabalho: bases de crenças e a representação de mundos possíveis.

Quando usamos *bases de crenças* nós assumimos que conjuntos de crenças podem ser inferidos de um pequeno conjunto de fórmulas. Uma base de crenças B_K representa um conjunto de crenças K se e somente se $K = Cn(B_K)$.

Na representação de *mundos possíveis*, nós não representamos as crenças por meio de fórmulas, mas sim através dos modelos (mundos) que satisfazem as crenças do agente. Denotamos por $[K]$ (algumas vezes $Mod(K)$) o conjunto de modelos onde todas as fórmulas de um conjunto de crenças K valem.

Quando o conjunto de crenças K é consistente, podemos ter três atitudes epistêmicas em relação a uma crença α [Gär88]:

i. Se $\alpha \in K$: α é aceita.

ii. Se $\neg\alpha \in K$: α é rejeitada.

iii. Se $\alpha \notin K$ e $\neg\alpha \notin K$: α é indeterminada.

As mudanças entre as diferentes atitudes epistêmicas definem o que chamamos operações em crenças. Denominamos por *expansão* quando passamos de uma indeterminação para a aceitação ou rejeição de uma crença. Na mudança inversão, quando passamos de uma aceitação ou rejeição, para indeterminação, realizamos uma *contração* de crenças. Por fim, quando mudamos nossa atitude de aceitação para rejeição, e vice-versa, realizamos uma operação de *revisão* de crenças.

3.2 Postulados de racionalidade

O Paradigma AGM define um conjunto de postulados de racionalidade, que guiam as operações sob o princípio da mudança mínima. Em síntese, mudar minimamente um conjunto de crenças significa que nenhuma crença deve ser adicionada ou removida se isso não for estritamente necessário ao sucesso da operação. Exceto pela operação de expansão, que é diretamente definida por $K + \alpha = Cn(K \cup \{\alpha\})$, as demais operações possuem um conjunto próprio de postulados de racionalidade.

3.2.1 Postulados para contração

Dado um conjunto de crenças K e uma crença α, os seis postulados básicos para a operação de contração são [Gär88]:

(K-1) $K - \alpha$ é um conjunto de crenças (*fecho*)

(K-2) $K - \alpha \subseteq K$ (*inclusão*)

(K-3) Se $\alpha \notin K$, então $K - \alpha = K$ (*vacuidade*)

(K-4) Se não $\vdash \alpha$, então $\alpha \notin K - \alpha$ (*sucesso*)

(K-5) Se $\alpha \in K$, então $K \subseteq (K - \alpha) + \alpha$ (*recuperação*)

(K-6) Se $\vdash \alpha \leftrightarrow \beta$, então $K - \alpha = K - \beta$ (*equivalência*)

O postulado **(K-1)** afirma que o resultado da contração de um conjunto de crenças é ainda um conjunto de crenças. O postulado **(K-2)** assegura que nenhuma nova fórmula é adicionada em uma operação de contração. Já o postulado **(K-3)** afirma que o conjunto de crenças permanece inalterado quando é contraído por uma crença que não possui. O postulado **(K-4)** assegura o sucesso da contração. O penúltimo postulado, **(K-5)**, indica que o conjunto de crenças original pode ser recuperado através da expansão pela crença a qual foi anteriormente contraído. Por fim, o postulado **(K-6)** certifica que duas crenças equivalentes, quando contraídas em um mesmo conjunto de crenças, geram resultados equivalentes.

Além dos seis postulados básicos, dois postulados adicionais tratam da operação de contração quando aplicada a conjunções de fórmulas:

(K-7) $K - \alpha \cap K - \beta \subseteq K - (\alpha \wedge \beta)$

(K-8) Se $\alpha \notin K - (\alpha \wedge \beta)$, então $K - (\alpha \wedge \beta) \subseteq K - \alpha$.

O postulado **(K-7)** afirma que crenças que estejam tanto em $K - \alpha$ quanto em $K - \beta$ devem pertencer a contração de K por $(\alpha \wedge \beta)$. O postulado **(K-8)** diz que se removemos α quando contraímos K por $(\alpha \wedge \beta)$, então qualquer crença que removemos de K quando fazemos $K - \alpha$ também deve ser removida quando o contraímos por $(\alpha \wedge \beta)$.

3.2.2 Postulados para revisão

Da mesma forma, dados K e α, os seis postulados básicos para operação de revisão são [Gär88]:

(K*1) $K * \alpha$ é um conjunto de crenças (*fecho*)

(K*2) $\alpha \in K * \alpha$ (*sucesso*)

(K*3) $K * \alpha \subseteq K + \alpha$ (*inclusão*)

(K*4) Se $\neg\alpha \notin K$, então $K + \alpha \subseteq K * \alpha$ (*preservação*)

(K*5) $K * \alpha = \mathcal{L}$ se e somente se $\vdash \neg\alpha$ (*consistência*)

(K*6) Se $\vdash \alpha \leftrightarrow \beta$, então $K * \alpha = K * \beta$ (*equivalência*)

O postulado **(K*1)** declara que a revisão de um conjunto de crenças resulta em um conjunto de crenças. O segundo postulado, **(K*2)**, assegura o sucesso da operação de revisão. O postulado **(K*3)** garante que nenhuma informação adicional é acrescida ao conjunto revisado. O postulado **(K*4)** (unido ao postulado (K*3)), afirma que quando a nova informação não é inconsistente com o conjunto de crenças, a revisão é equivalente a uma expansão. O postulado **(K*5)** diz que uma revisão gera um conjunto inconsistente apenas quando a crença é inconsistente. Finalmente, o postulado **(K*6)** certifica que duas crenças equivalentes resultam em conjuntos revisados equivalentes.

Assim como ocorre para contração, existem dois postulados adicionais aos seis básicos:

(K*7) $K * (\alpha \wedge \beta) \subseteq (K * \alpha) + \beta$.

(K*8) Se $\neg\beta \notin K * \alpha$, então $(K * \alpha) + \beta \subseteq K * (\alpha \wedge \beta)$

Os postulados **(K*7)** e **(K*8)**, unidos, afirmam que as interpretações que satisfazem $K*(\alpha\wedge\beta)$ são idênticas às interpretações que satisfazem simultaneamente $K * \alpha$ e β [Gär88].

3.2.3 Relações de identidade

Contração e revisão podem ser definidos um em função do outro por meio de duas identidades: *identidade de Levi* e *identidade de Harper* [Gär88]. Revisar um conjunto de crenças K com uma fórmula α equivale a contrair K com α e então adicionar α ao resultado:

$$K * \alpha = (K - \neg\alpha) + \alpha \qquad \textit{(identidade de Levi)}$$

Contrair um conjunto de crenças K com uma fórmula α equivale a revisar K com $\neg\alpha$ e então buscar os elementos comuns entre K e o resultado da revisão:

$$K - \alpha = (K * \neg\alpha) \cap K \qquad \textit{(identidade de Harper)}$$

Temos as seguintes equivalências entre as operações de contração e revisão, com respeito as identidades

Teorema 1. *[Gär88] Se $-$ é uma função de contração que satisfaz os postulados (K-1)-(K-4) e (K-6), então sua função de revisão associada $*$ satisfaz (K*1)-(K*6).*

Teorema 2. *[Gär88] Se $-$ é uma função de contração que satisfaz os postulados (K-1)-(K-4) e (K-6), então (a) se $-$ satisfaz (K-7), então sua função de revisão associada $*$ satisfaz (K*7) e (b) se $-$ satisfaz (K-8), então sua função de revisão associada $*$ satisfaz (K*8).*

Teorema 3. *[Gär88] Se $*$ é uma função de revisão que satisfaz os postulados (K*1)-(K*6), então sua função de revisão associada $-$ satisfaz (K-1)-(K-6).*

Teorema 4. *[Gär88] Se $*$ é uma função de revisão que satisfaz os postulados (K*1)-(K*6), então (a) se $*$ satisfaz (K*7), então sua função de contração associada satisfaz (K-7) e (b) se $*$ satisfaz (K*8), então sua função de contração associada satisfaz (K-8).*

3.3 Construção AGM

Em [AGM85], os autores propuseram uma construção para funções de contração/revisão capazes de satisfazer seus postulados de racionalidade: a construção *partial meet*.

Essa construção faz uso do conceito de *conjunto resíduo* para realizar a operação de contração. Um conjunto resíduo $K \perp \alpha$ consiste nos subconjuntos máximos de K que não implicam α.

Definição 9. [AGM85] Seja K um conjunto de fórmulas, α uma fórmula e Cn um operador de consequência, o *conjunto resíduo* $K\perp\alpha$ de K e α é definido como segue. Para todo conjunto X, $X \in K\perp\alpha$ se e somente se

1. $X \subseteq K$,
2. $\alpha \notin Cn(X)$,
3. para todo Y tal que $X \subset Y \subseteq K$, $\alpha \in Cn(Y)$.

A construção partial meet também utiliza o conceito de função de seleção, um modo de selecionar o melhor elemento de um conjunto resíduo de acordo com um critério de seleção pretendido.

Definição 10. [AGM85] Seja K um conjunto de fórmulas e α uma fórmula, uma *função de seleção* para K e α é uma função γ tal que

1. Se $K\perp\alpha \neq \emptyset$, então $\emptyset \neq \gamma(K\perp\alpha) \subseteq K\perp\alpha$,
2. Caso contrário, $\gamma(K\perp\alpha) = \{K\}$.

Uma função de contração partial meet é então definida como a interseção dos melhores elementos de um conjunto resíduo conforme escolhidos pela função de seleção. Formalmente uma contração *partial meet* é definida como segue.

Definição 11. [AGM85]
Seja K um conjunto de fórmulas, para qualquer fórmula α, um operador de contração *partial meet* sobre K e α, determinado por uma função de seleção γ é dado por

$$K -_\gamma \alpha = \bigcap \gamma(K\perp\alpha)$$

Como vimos na Seção 3.2.3, pela identidade de Levi podemos definir uma revisão em função de uma contração. Assim, podemos definir uma construção *partial meet* para revisão utilizando a construção operadores de contração dada pela Definição 11.

Definição 12. Seja K um conjunto de crenças, α uma nova crença e $-$ uma função de contração *partial meet*, uma *função de revisão partial meet* sobre K é dada por

$$K * \alpha = Cn(\{K - \alpha\} \cup \{\alpha\})$$

3.4 Equivalência entre construções e postulados

Em [AGM85], os autores mostram que é possível definir por meio de uma construção partial meet todas as operações de contração que satisfaçam os seis postulados básicos para contração. Denominamos esse resultado de *teorema de caracterização* para construção partial meet.

Teorema 5. *[AGM85] Seja $-$ uma função que, dado uma fórmula α, leva um conjunto de crenças K a um novo conjunto de crenças $K - \alpha$. Para todo conjunto de crenças K, a função $-$ é uma operação de contração partial meet se e somente se $-$ satisfaz os postulados de racionalidade (K-1)-(K-6) para contração.*

Como decorrência do Teorema 5, temos que a construção de operadores de revisão por meio da construção *partial meet* e identidade de Levi completamente caracteriza todas as possíveis operações que satisfazem os seis postulados básicos de revisão.

Teorema 6. *[AGM85] Seja $*$ uma função que, dado uma fórmula α, leva um conjunto de crenças K a um novo conjunto de crenças $K * \alpha$. Para todo conjunto de crenças K, a função $*$ é uma operação de revisão partial meet se e somente se $*$ satisfaz os postulados de racionalidade (K*1)-(K*6) para revisão.*

Capítulo 4

Revisão de Crenças no Reparo de Especificações de Sistemas

Em verificação formal, descrevemos sistemas computacionais por meio de uma linguagem formal, a fim de representá-lo de modo abstrato. Utilizamos este modelo para checá-lo frente um conjunto de propriedades desejadas. Essa verificação permite que detectemos falhas em um sistema antes de sua implementação de fato.

Assumimos aqui que nossos sistemas são modelados por sistemas de transição rotulados e então verificados frente a conjuntos de propriedades expressas em lógica temporal. Dado um resultado negativo do processo de verificação de modelos, o projetista deve então adaptar o modelo do sistema para reparar o erro encontrado.

A representação do sistema como um LTS pode ser vista de dois modos: (1) o modelo descreve exatamente (e completamente) o comportamento esperado do sistema, que estados este deve possuir e como devem ocorrer a transição entre eles; ou (2) o modelo é uma instância de uma especificação descrita em uma linguagem de mais alta ordem, sendo visto como um possível modelo concreto capaz de satisfazer as propriedades esperadas.

No primeiro caso, o reparo necessariamente ter foco no LTS, dado que ele completamente define o sistema. Denominamos esse como *problema do reparo de modelos*. No segundo caso, o modelo é apenas produto de uma descrição errada de uma especificação, e o foco da alteração deve ser as propriedades (fórmulas) que compõe tal especificação. Denominamos esse problema de *revisão de especificações*.

A principal questão é se é possível definir revisão de especificações em termos de revisão de modelos, e vice-versa. Como consequência, poderíamos definir revisão de crenças sobre conjuntos de fórmulas temporais utilizando operações de reparo de modelos, bem como poderíamos atestar que abordagem clássicas de revisão de crenças poderiam ser utilizadas para o reparo de modelos.

Nesta seção exploramos a correspondência entre as duas diferentes perspectivas do problema. Discutiremos questões relativas a correspondência entre reparo de modelos e revisão de especificações. Mostramos que em certas situações essas abordagens não são equivalentes, o que indica que pode não existir uma única solução para ambas perspectivas.

Apesar de parecer um resultado intuitivo, trabalhos como [CBSK12], [GW10], [SW07] e [ZD08] foram desenvolvidos sem apropriadamente destacar a diferença entre as possíveis interpretações do problema. Isso pode induzir a crença de que para lógicas temporais, modificar conjuntos de fórmulas e modificar modelos geram o mesmo resultado, como ocorre para lógica clássica. Evidenciamos aqui que tal interpretação deve ser guiar a escolha que qual abordagem escolher.

4.1 Revisão de especificações

Na abordagem de revisão de especificações, o comportamento esperado de um sistema é especificado por um conjunto de fórmulas temporais, que representa as propriedades esperadas do sistema

durante seu funcionamento. Para efeitos de ilustração, assumimos tal especificação será representada por um conjunto de fórmulas CTL, contudo o princípio continua valido para outras lógicas temporais, como a lógica LTL.

Uma especificação de sistema é composta de fórmulas que descrevem propriedades como invariantes, *safety* e *liveness*. Esse conjunto de fórmula é tomado então como nosso conjunto inicial de crenças sobre o sistema. Como em revisão de crenças clássicas, quando uma informação desejada (no nosso caso, a propriedade checada via verificação formal) é inconsistente com o conjunto de crenças, devemos adaptar nossas crenças com o objetivo de incorporar a nova informação.

Suponha, por exemplo, que temos o seguinte conjunto de fórmulas LTL como propriedades que especificam um sistema

$$K = Cn(\{\mathrm{EF}p, \mathrm{AG}(p \rightarrow q)\}).$$

Dizemos que K é consistente com uma fórmula ϕ se existe um modelo que satisfaz todas as fórmulas em $K \cup \phi$. O conjunto K é consistente com $\phi_1 = q$, todo modelo que cujo estado inicial satisfaz p e q, satisfaz $K \cup \phi_1$. Contudo K e $\phi_2 = \mathrm{G}\neg q$ são inconsistentes, todo modelo que satisfaz K, p eventualmente vale, e assim também q, logo não pode ser o caso nestes modelos que globalmente temos $\neg q$.

O objetivo da revisão de especificação é maximalizar a preservação das crenças originais. Este princípio tem correspondência direta com a construção *partial meet*, baseada em conjuntos maximais consistentes.

Neste exemplo, o conjunto resíduo de K e $\neg\mathrm{G}\neg q$ é uma coleção de conjuntos $K\perp(\neg\mathrm{G}\neg q) = \{X, Y_0, Y_1, Y_2, Y_3, ...\}$, onde

1. $X = Cn(\{\mathrm{AG}(p \rightarrow q)\})$, e
2. $Y_i = Cn(\{\mathrm{AF}p\} \cup \{\mathrm{AX}^n(p \rightarrow q) \mid n \neq i\})$.

onde $\mathrm{AX}^n\psi$ representa $\underbrace{\mathrm{AXAX} \ldots \mathrm{AX}}_{n}\psi$.

Baseado em $K\perp(\neg\mathrm{G}\neg q)$, temos um número infinito de possibilidades para a revisão de especificação K de modo a incorporar a nova crença $\mathrm{G}\neg q$.

Relativamente poucos trabalhos abordam revisão de crenças sobre lógicas não clássicas. A principal questão é que muitos dos resultados clássicos não se aplicam a lógica que não satisfazem propriedades como monotonicidade ou compacidade. Lógicas temporais em geral não satisfazer compacidade, e apesar de termos uma solução para o exemplo apresentado, não há garantia que existam funções de revisão para qualquer conjunto de fórmulas.

Para alguns casos específicos, como conjuntos de crenças finitos, podemos corretamente aplicar o paradigma AGM para revisão de fórmulas. Esta restrição será suficiente para a questão abordada neste capítulo.

4.2 Revisão de modelos

O outro modo de descrever um sistema é explicitamente representado sua estrutura de funcionamento, seus estados e suas transições.

O reparo de modelos consiste em, assumindo a estrutura de um modelo como conjunto inicial de crenças, como reparar este modelo mudando minimamente sua estrutura, de modo a preservar o máximo possível das características iniciais pretendidas.

Este critério de minimalidade pode ser um fator importante para o projetista. Em muitas aplicações, a adição de um estado pode, por exemplo, demandar o desenvolvimento de novos componentes de um sistema, e consequentemente implicar no aumento do custo final.

Em [GW10], apresentamos um arcabouço inicial para o reparo estrutural de modelos baseado em revisão de crenças. Propomos o uso de certos princípios de revisão de crenças para realizar escolhas

de mudanças que minimamente alteram um modelo. Essa abordagem será tratada em detalhes no Capítulo 6, para esta seção será necessária apenas a intuição sobre mudanças estruturais mínimas.

Neste arcabouço para o reparo de modelos, uma modificação é formalmente definida como uma composição de operações primitivas, conforme originalmente proposto por [ZD08].

PU1: Adicionar um par a relação R

PU2: Remover um par da relação R

PU3: Alterar o rótulo de um estado

PU4: Adicionar um novo estado

PU5: Remover um estado isolado

Essas operações são utilizadas para modificar um modelo gerando diversas possibilidades de reparos para que este seja consistentes com a propriedade desejada. Os modelos resultantes desses possíveis reparos são então comparados quanto a sua similaridade estrutural: diferença entre estados adicionados ou removidos, estados que mudaram de rotulação, etc. Os modelos são então classificados em uma ordem de proximidades e aqueles mais similares ao modelo original segundo essa ordem, são escolhidos como resultado do reparo.

A intuição é que a modificação deve ser mínima em relação a todas a possíveis modificações, isto é, não deve existir nenhum modelo modificado consistente com a propriedade desejada, com menos mudanças estruturais com respeito ao modelo original.

4.3 Diferença entre revisão de modelos e revisão de especificações

Para lógica proposicional, a revisão de crenças produz resultados equivalentes quando aplicados a conjuntos de fórmulas ou a modelos semânticos [Gro88]. Isso significa que podemos definir revisão de conjuntos de fórmulas em termos de uma relação de proximidade entre seus possíveis modelos, e vice-versa.

Se tal definição for possível no contexto das lógicas temporais, seria possível por exemplo definir revisão de fórmulas temporais por meio dos trabalhos existente em revisão de modelos. Caso não seja possível, isso indica a necessidade de evoluir as duas abordagens separadamente, sendo cada uma foco de uma demanda específica.

Nesta seção mostramos por meio de exemplos que as duas abordagem produzem resultados disjuntos, sendo, portanto, uma indicação que a revisão de especificações e a revisão de modelos não podem ser interdefiníveis.

4.3.1 Diferenças entre os dois resultados

Seja p_1, p_2 e p_3 os únicos símbolos proposicionais de um dado sistema. Suponha que acreditamos que no nosso sistema as seguintes propriedades são verdadeiras: p_1 vale globalmente; se vale $\neg p_1$ em algum estado, existe um caminho a partir de um estado seguinte onde p_2 vale globalmente; o mesmo vale para p_3; por fim, p_1, p_2 e p_3 são mutualmente exclusivos.

Seja K nosso conjunto de crenças baseado nas propriedades acima. Definimos K como

$$K = Cn(\{\mathrm{AG}p_1, \mathrm{EX}\neg p_1 \rightarrow \mathrm{EXAG}p_2, \mathrm{EX}\neg p_1 \rightarrow \mathrm{EXAG}p_3, \phi_{mutex}\})$$

onde

$$\phi_{mutex} \equiv \neg\mathrm{EF}(p_1 \wedge p_2) \wedge \neg\mathrm{EF}(p_1 \wedge p_3) \wedge \neg\mathrm{EF}(p_2 \wedge p_3).$$

Suponha que percebemos que p_2 ou p_3 deve necessariamente ser satisfeitos em um estado seguinte ao inicial

$$\mathrm{EX}(p_2 \vee p_3).$$

Esta nova informação é inconsistente como nossa base de crenças atual K, então desejamos revisar K por $\mathrm{EX}(p_2 \vee p_3)$. Neste exemplo, também assumimos que a propriedade de exclusão mutua deve ser mantida, assim nossa meta é assegurar

$$\psi \equiv \mathrm{EX}(p_2 \vee p_3) \wedge \phi_{mutex}$$

Para uma revisão de especificação $K * \psi$, desejamos encontrar os subconjuntos maximais de K que são consistentes com a nova propriedade ψ. Neste exemplo, $\mathrm{EX}(p_2 \vee p_3)$ é inconsistente com $\mathrm{AX}p_1$, inferido a partir de $\mathrm{AG}p_1$, assim devemos desistir desta última informação em nossa base de crenças. O único subconjunto maximal consistente $X \subset K\perp\neg\psi$ é dado então por

$$\begin{aligned} X = Cn(\{\, & p_1, \mathrm{EX}p_1, \mathrm{AXAX}p_1, \mathrm{AXAXAX}p_1, \ldots, \phi_{mutex}, \\ & \mathrm{EX}\neg p_1 \rightarrow \mathrm{EXAG}p_2, \mathrm{EX}\neg p_1 \rightarrow \mathrm{EXAG}p_3 \,\}) \end{aligned}$$

Como $K\perp\neg\psi$ é um conjunto unário, existe apenas uma possível escolha para γ e assim o único resultado possível para a revisão de K por ψ é dado por

$$K * \psi = Cn(X \cup \{\mathrm{EX}(p_2 \vee p_3) \wedge \phi_{mutex}\})$$

Do ponto de vista da revisão de modelos, nossas crenças são compostas pelo conjunto $\mathcal{K}$ de todos os possíveis modelos que satisfazem o conjunto de propriedades K. Dado a restrição ϕ_{mutex}, todo modelo para K tem estados rotulados com no máximo um entre p_1, p_2 e p_3. Dado a fórmula $\mathrm{AG}p_1$, todos os possíveis modelos para K são bissimilares ao modelo M na Figura 4.1.

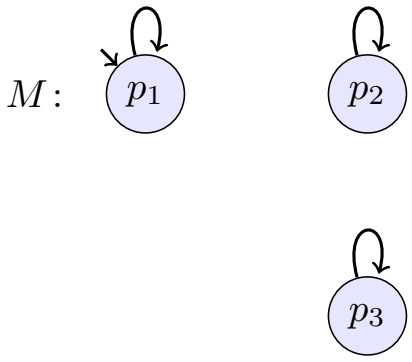


Figura 4.1: *Modelo do conjunto inicial de crenças.*

De acordo com o objetivo do reparo de modelos, precisamos achar as mudanças estruturais mínimas nos modelos que satisfazem as propriedades iniciais, de modo a produzir modelos que satisfazem $\mathrm{EX}(p_2 \vee p_3) \wedge \phi_{mutex}$. Isto resulta no seguinte conjunto de modelos revisados

$$\mathcal{K}' = \mathcal{K} \circ_c (\mathrm{EX}(p_2 \vee p_3) \wedge \phi_{mutex})$$

Cada modelo $M \in \mathcal{K}$ pode ser reparado com apenas uma adição de transição entre um estado p_1 (com um loop) para um estado que satisfaz p_2 ou p_3, ambos com um loop. Qualquer modificação adicional é considerada redundante. Figura 4.2 mostra exemplos de possíveis reparos para o modelo.

O resultado da revisão $\mathcal{K}'$ contém apenas modelos que são estruturalmente semelhantes ao modelo original, e que pode vir a demandar menos esforços de um projetista de sistema em adaptar o modelo original.

A principal diferença surge quando relacionamos as duas abordagens. Por um lado, não existe qualquer modelo em $\mathcal{K}'$ capaz de satisfazer as fórmulas resultantes da revisão de especificação

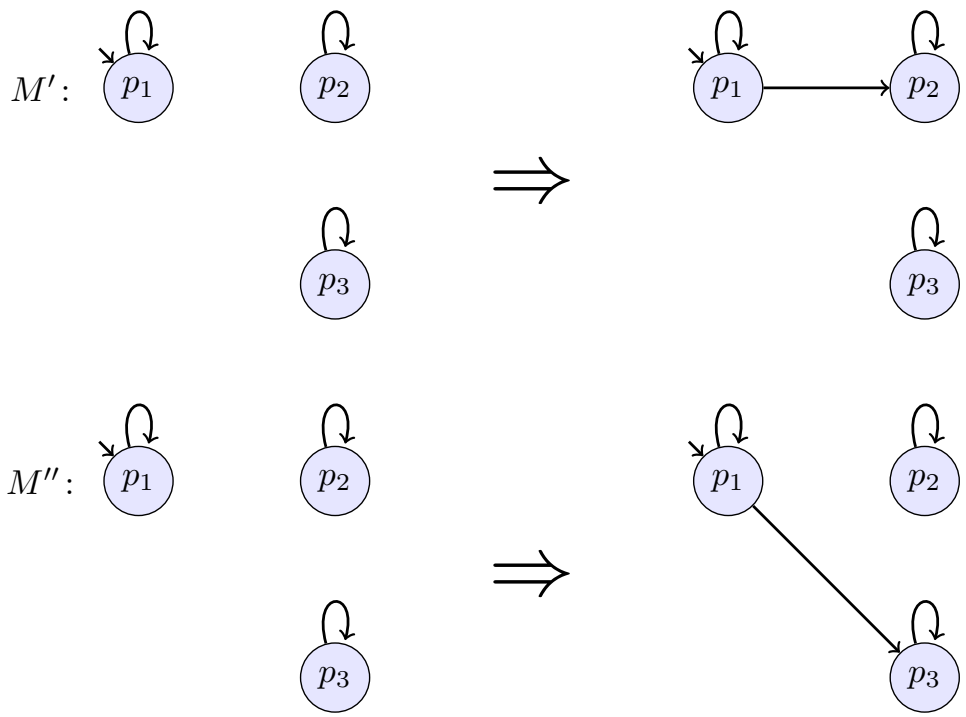


Figura 4.2: *Soluções para o reparo $M \in \mathcal{K}$.*

K'. Todo modelo em $\mathcal{K}'$ pode satisfazer apenas uma fórmula entre $\text{EX}\neg p_1 \rightarrow \text{EXAG}p_2$ e $\text{EX}\neg p_1 \rightarrow \text{EXAG}p_3$, já que apenas uma transição foi adicionada. Este resultado implica uma não maximalidade da preservação do conjunto original de fórmulas.

Por outro lado, não existe modelo de K' que possa ser usado como resultado para a revisão de modelos. Para satisfazer $\text{EX}\neg p_1 \rightarrow \text{EXAG}p_2$ e $\text{EX}\neg p_1 \rightarrow \text{EXAG}p_3$, todo modelo de K' contém pelo menos duas transições que não existiam anteriormente. A Figura 4.3 mostra um exemplo de modelo para K', não minimal com respeito a $\mathcal{K}$, dado que é possível satisfazer a propriedade com menos modificações (ver Figura 4.2).

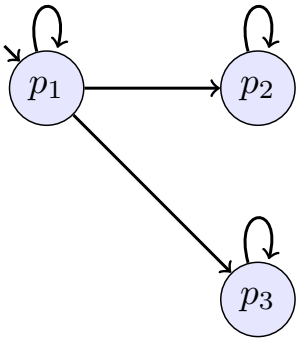


Figura 4.3: *Solução não minimal para a revisão de modelos.*

Com base nesse exemplo, podemos ver que não existe modelo para K' que preserva a similaridade estrutura como pretendido pela revisão de modelos. Do mesmo modo, não existe modelo em $\mathcal{K}'$ que preserva maximamente o conjunto de propriedades que compõe a especificação, como pretendido pela revisão de especificações.

4.3.2 Implicações da diferenciação

Este resultado mostra que, diferente da abordagem tradicional de revisão de crenças sobre lógica proposicional, não é o caso que a revisão de crenças sobre o formalismo da lógica CTL pode ser

realizada sintaticamente e semanticamente obtendo resultados equivalentes.

A maioria das técnicas existentes para CTL (por exemplo, verificação de modelos ou tableaux) são realizados sobre modelos devido a inúmeras propriedades "bem comportadas" neste formalismo. Dado que nosso resultado evidencia a fronteira entre modelos e fórmulas, a literatura ainda é carente de resultados no raciocínio sintático sobre fórmulas CTL.

De fato, existe casos onde as duas abordagens têm interseção em seus resultados. Por o exemplo da Seção 4.3.1, se nós tivéssemos removidos a restrição ϕ_{mutex}, a abordagem de reparo de modelos iria produzir pelo menos um modelo na qual o resultado da revisão de especificação seria completamente satisfeito (um modelo com um estado inicial com todas as proposições). Contudo, não está claro se existe uma classe de operadores de revisão onde as duas abordagens são completamente compatíveis.

Adicionalmente, os esforços de buscar tal classe de equivalência não parecem práticos. Isso pode requerer uma restrição forte nos tipos de conjuntos de fórmulas usados na especificação, talvez restringindo o conjunto apenas para fórmulas com operador X e uma correspondência um-para-um entre fórmulas e transições do modelo. Outra possibilidade é a desistir da restrição de proximidade estrutural para a revisão do modelo, contudo isso pode contrastar com necessidades do projetista do sistema, comprometendo a usabilidade da abordagem em aplicações reais.

Este resultado mostra-se importante para o desenvolvimento de uma solução para o reparo de sistemas, onde a evolução deste pode ser melhorada se a separação entre especificação e modelo é tomada em consideração.

Capítulo 5

Revisão de Bases de Crenças Temporais

Neste capítulo investigamos quando a construções tradicionais podem ser aplicadas a lógicas temporais. Inicialmente, investigamos especificamente o problema de como definir operadores de contração baseados na construção *partial meet* [AGM85] para a lógica LTL. Mostramos que, dado um conjunto qualquer de fórmulas LTL K e uma fórmula LTL α, verificar a existência de conjunto resíduo de K e α é no caso geral indecidível.

O teste de vacuidade de conjuntos resíduos é um passo crucial na construção partial meet, e esse resultado nos leva a explorar que conjuntos de restrições podem ser aplicadas de modo que o conjunto resíduo possa ser computado. Mostramos que o resultado também é extensível a outras lógicas além da LTL. Em especial mostramos que a indecidibilidade também ocorre para as lógicas CTL e PDL [FL79, HKT00].

Para demonstrar que o problema de verificar a existência de conjunto resíduos para LTL é no caso geral indecidível, faremos uma redução do problema da vacuidade de linguagens aceitar por um tipo especial de máquina de Turing, denominado *autômatos linearmente limitados*.

5.1 Autômatos linearmente limitados

Um *autômato linearmente limitado* (ALL) é um tipo de máquina de Turing onde a cabeça de leitura da máquina não pode mover-se para porções da fita além daquela que contém a entrada [Sip05]. Proposto por [Myh60], ALL aproximam máquinas de Turing de computadores reais, modelando seu processo de computação com memória limitada.[1]

Definição 13. Um *autômato linearmente limitado* (ALL) é uma tupla $\langle Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r \rangle$, onde

1. Q é um conjunto finito de estados,
2. Σ é um finito de símbolos de entrada (alfabeto de entrada),
3. Γ é um finito de símbolos de fita (alfabeto de fita), onde $\Sigma \subseteq \Gamma$,
4. $\delta : Q' \times \Gamma \to Q \times \Gamma \times \{L, R\}$ é a função de transição, onde $Q' = Q \setminus \{q_a, q_r\}$
5. $q_0 \in Q$ é o estado inicial,
6. $q_a \in Q$ é o estado de aceitação, e
7. $q_r \in Q$ é o estado de rejeição, onde $q_a \neq q_r$.

e δ é definido de modo que o ALL não possa mover sua cabeça de leitura fora das posições ocupadas pela cadeia de entrada[2].

[1] As definições 13-16 são baseadas naquelas apresentadas em [Sip05] para máquinas de Turing, com pequenas adaptações para o contexto de ALL.

[2] Isso pode ser feito reservando dois símbolos de Σ para representar marcas de inicio e fim da entrada, de modo que a função de transição sobrescrever as marcas com símbolos diferentes destas, mover para a esquerda da marca de inicio, ou mover para a direita da marca de fim.

A Figura 5.1 ilustra o diagrama de estados de um ALL M formalmente definido como $M = \langle\{q_0, q_a, q_r\}, \{a, b\}, \{a, b\}, \delta, q_0, q_a, q_r\rangle$ onde $\delta(q_0, a) = (q_a, a, R)$, $\delta(q_0, b) = (q_r, b, R)$ e indefinido caso contrário.

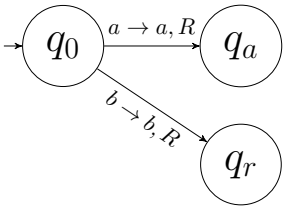


Figura 5.1: *Diagrama de um ALL.*

Durante a computação de um ALL, mudanças podem acontecer no conteúdo da fita, na posição da cabeça de leitura-escrita ou no estado de controle do autômato. Esses itens definem *configurações* de um ALL. Tais configurações são representados por uma única cadeia como descrita na Definição 14.

Definição 14. Seja $M = \langle Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r\rangle$ um ALL e $C = c_1c_2...c_m$ uma cadeia sobre o alfabeto $Q \cup \Gamma$, dizemos que C é uma configuração válida de M se e somente se $c_i \in Q$, para algum $1 \leq i \leq m$, e $c_j \in \Gamma$, para todo $i \neq j$.

Por exemplo, q_0aa e aq_aabb são configurações válidas de M da Figura 5.1, enquanto q_0q_rb ou $abab$ não.

Um passo em uma computação de um ALL é definido como uma transição de uma configuração para outra de acordo com as regras especificadas pela função de transição.

Definição 15. Seja $M = \langle Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r\rangle$ um ALL e C_i e C_j duas configurações validas de M, dizemos que C_i leva a C_j se, para $a, b, c \in \Gamma$, $u, v \in \Gamma^*$ e $q_i, q_j \in Q$, uma das seguintes condições é satisfeita:

1. $C_i = uq_iav$, $C_j = ubq_jv$ e $\delta(q_i, a) = (q_j, b, R)$.
2. $C_i = uaq_ibv$, $C_j = uq_jacv$ e $\delta(q_i, b) = (q_j, c, L)$.

A configuração inicial de um ALL M com entrada w é q_0w. Uma *configuração de aceitação* é uma configuração na qual o símbolo de estado é q_a. Uma *configuração de rejeição* é uma configuração na qual o símbolo de estado é q_r. Configurações de aceitação e rejeição são *configurações de parada* e não levam a nenhuma outra configuração.

Definição 16. Seja M um ALL e w uma cadeia de entrada para M, uma *computação* de M sobre w é uma sequência de configurações $C_1, C_2, ..., C_l$ onde

1. C_1 é uma configuração inicial,
2. Cada C_i leva à C_{i+1}.

Se C_l é uma configuração de aceitação, dizemos que M *aceita* w. Se C_l é uma configuração de rejeição, dizemos que M *rejeita* w. Em ambos os casos, dizemos que M *para* sobre a entrada w.

Uma propriedade interessante de ALL é que, em contraste com o caso geral para máquinas de Turing, o *problema da aceitação* para ALL é decidível. Devido ao número limitado de possíveis configurações para uma dada entrada, decidir quando um ALL aceita ou não uma entrada pode ser feito em um número finito de passos, uma vez que, para um ALL M com q estados e g símbolos de fita, existe exatamente qng^n configurações distintas para uma entrada w de tamanho n. Assim, se M aceita w sua computação deve levar menos do que qng^n passos de computação para alcançar configurações de aceitação.

Contudo, outros problemas indecidíveis para máquinas de Turing continuam indecidíveis para ALL. É o caso, por exemplo, do *problema da vacuidade para ALL*, que consiste em determinar quando um dado ALL aceita qualquer entrada.

Definimos a *linguagem* de ALL M como o conjunto $L(M) = \{w \in \Sigma^* \mid M \text{ aceita } w\}$. Definimos E_{ALL} como o conjunto de todos os ALL que possuem linguagem vazia,

$$E_{ALL} = \{\langle M \rangle \mid M \text{ é um ALL onde } L(M) = \emptyset\}$$

O teste de pertinência sobre E_{ALL} é um problema indecidível[3].

Na próxima seção, estabelecemos a indecidibilidade da construção partial meet sobre conjuntos de fórmulas temporais, baseado em uma redução do problema da vacuidade de ALL para este problema. Mostramos que se nosso problema fosse decidível, poderíamos decidir se um dado ALL pertence ou não a E_{ALL}, uma contradição.

5.2 Indecidibilidade de contrações partial meet para LTL

Nesta seção, mostraremos que é impossível definir um operador de contração partial meet para conjuntos de crenças LTL dado a indecidibilidade do *problema da vacuidade de* $K \perp \alpha$.

Definição 17. Seja K um conjunto de fórmulas LTL e $\alpha \in \mathcal{L}_{LTL}$ uma fórmula LTL, o *problema da vacuidade de* $K \perp \alpha$, $E_{K \perp \alpha}$, consiste em determinar se existe um conjunto resíduo para K e α. Formalmente $E_{K \perp \alpha}$ é definido como

$$E_{K \perp \alpha} = \{\langle K, \alpha \rangle \mid K \perp \alpha = \emptyset\},$$

onde $K \perp \alpha = \emptyset$ se e somente se $\langle K, \alpha \rangle \in E_{K \perp \alpha}$.

Para provar que $E_{K \perp \alpha}$ é indecidível, nós apresentamos uma redução do problema E_{ALL} usando o conceito de fórmulas ALL-*equivalentes*.

5.2.1 Construção de fórmulas ALL-*equivalentes*

Uma fórmula ALL-*equivalente* ϕ_n^{M} é uma fórmula LTL que *codifica* uma autômato linearmente limitado M no sentido que cada modelo que satisfaz ϕ_n^{M} descreve uma computação de M para uma cadeia de entrada com n símbolos.

Para simplificar a notação, primeiro definimos uma função $\sigma : \Sigma^* \to \text{LTL}$ que converte cadeias arbitrárias em fórmulas LTL,

$$\sigma(c_0 c_1 ... c_m) = \bigwedge_{i=0}^{m} \mathrm{X}^i c_i, \text{ where } c_i \in \Sigma^*.$$

Por exemplo, $\sigma(abca) = a \wedge \mathrm{X}b \wedge \mathrm{XX}c \wedge \mathrm{XXX}a$, onde $\Sigma = \{a, b, c\}$.

Seja $M = \langle Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r \rangle$ um ALL, $\#$ um símbolo que não pertence a Q ou Γ, e $AP = Q \cup \Gamma \cup \{\#\}$ um conjunto de átomos proposicionais, construímos uma fórmula LTL ϕ_n^{M} sobre os átomos AP, baseada no conjunto de todas as possíveis configurações $C_1, C_2, ..., C_l$ de M e todas as entradas com exatamente n símbolos. O principal objetivo é capturar aspectos básicos da computação de M, como configurações iniciais e de parada, e suas transições passo-a-passo.

Definição 18. Seja M um ALL e $C_1, C_2, ..., C_l$ o conjunto de todas as possíveis configurações de M com tamanho $n + 1$. Definimos Π como o menor conjunto de fórmulas LTL tais que

1. $\bigvee_{\beta \in \Delta} \beta \in \Pi$, onde $\Delta = \{\sigma(\#q_0 w \#) \mid w \in \Sigma^* \text{ e } w \text{ tem exatamente } n \text{ símbolos }\}$ (I)

[3] Uma prova didática deste resultado pode ser visto em [Sip05].

2. Para cada C_i em $C_1, C_2, ..., C_l$,

- Se $C_i = uqv$, onde $u, v \in \Gamma^*$, $q \in \{q_a, q_r\}$, então

$$G(\sigma(\#uqv\#) \rightarrow \sigma(\#uqv\#uqv\#)) \in \Pi \tag{H}$$

- Se $C_i = uqav$ e $\delta(q, a) = (r, b, R)$, onde $a, b \in \Gamma,\ u, v \in \Gamma^*$, $q, r \in Q$, então

$$G(\sigma(\#uqav\#) \rightarrow \sigma(\#uqav\#ubrv\#)) \in \Pi \tag{R}$$

- Se $C_i = ubqav$ e $\delta(q, a) = (r, c, L)$, onde $a, b, c \in \Gamma,\ u, v \in \Gamma^*$, $q, r \in Q$, então

$$G(\sigma(\#ubqav\#) \rightarrow \sigma(\#ubqav\#urbcv\#)) \in \Pi \tag{L}$$

3. $G((\bigvee_{p \in AP} p) \wedge (\bigwedge_{p \in AP} (p \rightarrow (\bigwedge_{q \in AP \setminus \{p\}} \neg q\ )))) \in \Pi$ (X)

A fórmula ϕ_n^{M} é então definida como a conjunção de todas as fórmulas em Π

$$\phi_n^{\mathrm{M}} = \bigwedge_{\psi \in \Pi} \psi.$$

e denominamos ϕ_n^{M} de fórmula *ALL-equivalente* a M com entrada de tamanho n.

A cláusula (I) assegura que cada modelo $\mathcal{M}$ que satisfaça ϕ_n^{M} codifique uma configuração inicial válida em sua sequência inicial de estados. A cláusula (H) assegura que se C_i é uma configuração de parada, então $\sigma(\#C_i\#)$ essa configuração passa a ocorrer infinitamente em $\mathcal{M}$, simulando assim a não troca de configurações. A cláusulas (R) e (L) asseguram que, se a configuração C_i leva a C_j, todos os fragmentos de caminho de $\mathcal{M}$ que tornam a fórmula $\sigma(\#C_i\#)$ verdadeira são imediatamente seguidos por uma sequência de estados que satisfazem $\sigma(\#C_j\#)$. Finalmente, a cláusula (X) assegura que apenas um elemento do conjunto de proposições seja válida em cada estado[4].

Suponha, por exemplo, que ϕ_2^{M} é uma fórmula ALL-equivalente ao autômato M ilustrado na Figura 5.1 sobre entradas com 2 símbolos, seguindo a construção dada na Definição 18. Pela cláusula (I), todos os modelos que satisfazem ϕ_2^{M} devem satisfazer

$$\sigma(\#q_0aa\#) \vee \sigma(\#q_0ab\#) \vee \sigma(\#q_0ba\#) \vee \sigma(\#q_0bb\#)$$

Se um modelo $\mathcal{M}$ satisfaz ϕ_2^{M}, sua sequência inicial de estados deve codificar uma configuração inicial válida de M. A Figura 5.2 ilustra uma possível sequência inicial de estados para $\mathcal{M}$.

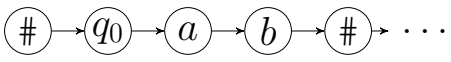


Figura 5.2: *Possível sequência inicial de estados de $\mathcal{M}$*

Dado que $\mathcal{M}$ também satisfaz (R) em ϕ_2^{M}, que $\sigma(\#q_0ab\#)$ vale no estado inicial de $\mathcal{M}$ e que $\delta(q_0, a) = (q_a, a, R)$ é uma transição em M, o modelo $\mathcal{M}$ deve também satisfazer $\sigma(\#q_0ab\#aq_ab\#)$ em seu estado inicial. Figura 5.3 ilustra a sequência seguinte de estados em $\mathcal{M}$.

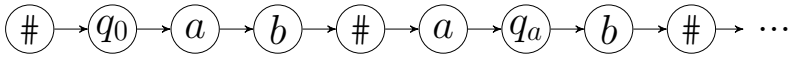


Figura 5.3: *Sequência seguinte de estados de $\mathcal{M}$*

[4]Essa cláusula é importante nas próximas seções construir a equivalência entre modelos LTL e histórico de computação de uma ALL.

A configuração aq_ab é uma configuração de aceitação de M. Dado que $\mathcal{M}$ satisfaz as cláusulas (H) em ϕ_2^{M}, essa sequência de estados deve seguir infinitamente em $\mathcal{M}$. Isso pode ser alcançado pela definição de um laço neste conjunto de estados

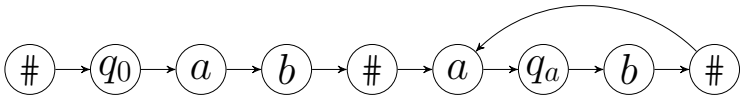


Figura 5.4: *Diagrama de estados de um modelo $\mathcal{M}$ para ϕ_2^M.*

O modelo da figura Figura 5.4 é um modelo ϕ_2^{M} que codifica um histórico de computação de aceitação para o ALL M com a cadeia de entrada ab.

5.2.2 Indecidibilidade do problema $E_{K\perp\alpha}$

Mostramos no Lema 7 que a aceitação de uma entrada por um ALL M está diretamente relacionada a satisfação da fórmula $\phi_n^{\mathrm{M}} \wedge \mathrm{F}q_a$.

Lema 7. *Seja $M = \langle Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r\rangle$ um ALL e w uma cadeia de entrada para M com n símbolos,*

$$M \text{ aceita } w \text{ se e somente se } \phi_n^M \wedge \mathrm{F}q_a \text{ é satisfazível.}$$

Demonstração. *Esquerda para direita.* Se M aceita a entrada w, então existe uma sequência de configurações $C_1, C_2, ..., C_l$ que M passa para aceitar w. Seja $\mathcal{C} = \#C_1\#...\#C_l\#$ a concatenação de todas as configurações $C_1, C_2, ..., C_l$, cada qual separada pelo símbolo $\# \notin Q \cup \Gamma$. Construímos um modelo $\mathcal{M} = \langle AP, S, s_0, R, L\rangle$ a partir de $\mathcal{C}$ tal que

- $AP = \Gamma \cup Q \cup \{\#\}$,
- $S = \{s_0, s_1, ..., s_{|\mathcal{C}|}\}$,
- $R = \{(s_i, s_{i+1}) \mid 0 \leq i < |\mathcal{C}|\} \cup \{(s_{|\mathcal{C}|}, s_{|\mathcal{C}|-(n+1)})\}$
- para todo $s_i \in S$, $L(s_i) = \{c_i\}$, onde c_i é o i-ésimo símbolo de $\mathcal{C}$.

Dado que M aceita w, temos que q_a é um símbolo de $\mathcal{C}$. Por construção, $s_i \in L(q_a)$ para algum $s_i \in S$ que pode ser alcançado a partir do estado inicial, logo vale que $\mathcal{M} \models \mathrm{F}q_a$. Seja n o número de símbolos de w, como C_1 é uma configuração inicial válida, o modelo $\mathcal{M}$ satisfaz a cláusula (I) de ϕ_n^{M}. Por construção, cada estado de $\mathcal{M}$ é relacionado a um único símbolo proposicional, logo $\mathcal{M}$ satisfaz a cláusula (X) de ϕ_n^{M}. Dado que cada C_{i+1} segue legitimamente de C_i, para cada estado de $\mathcal{M}$, ou $\sigma(\#C_i\#)$ não vale no estado ou é verdade nele que $\sigma(\#C_i\#C_{i+1}\#)$. Em ambos os casos, as implicações das cláusulas (R) e (L) são globalmente satisfeitas em $\mathcal{M}$. Finalmente, dado que C_l é uma configuração de aceitação, a transição $(s_{|\mathcal{C}|}, s_{|\mathcal{C}|-(n+1)}) \in R$ assegura a satisfazibilidade da cláusula (H). Portanto, $\mathcal{M} \models \phi_n^{\mathrm{M}}$ e assim $\phi_n^{\mathrm{M}} \wedge \mathrm{F}q_a$ é satisfazível.

Direita para esquerda. Se $\phi_n^{\mathrm{M}} \wedge \mathrm{F}q_a$ é satisfazível, então existe um modelo $\mathcal{M} = \langle AP, S, s_0, R, L\rangle$ tal que $\mathcal{M} \models \phi_n^{\mathrm{M}} \wedge \mathrm{F}q_a$. Como $\mathcal{M} \models \mathrm{F}q_a$, existe em $\mathcal{M}$ um caminho de computação $\pi = s_0 \to s_1 \to \cdots \to s_j \to \cdots \to s_k \to \ldots$ tal que $s_i \in L(q_a)$ para algum $i \geq 0$. Seja s_j o primeiro estado desse caminho tal que $s_j \in L(q_a)$ e s_k o primeiro estado após s_j tal que $s_k \in L(\#)$. Seja $\mathcal{C} = c_1c_2...c_k$ uma cadeia tal que $s_i \in L(c_i)$, para $1 \leq i \leq k$. Dividindo $\mathcal{C}$ de acordo com os delimitadores $\#$, obtemos a sequência de configurações $C_1, C_2, ..., C_l$.Resta mostrar que tal sequência é um histórico de computação de aceitação de M para alguma entrada com n símbolos. Por construção de ϕ_n^{M}, cada C_{i+1} dele seguir legitimamente de C_i, dado que as cláusulas (R) e (L) são definidas a partir da função δ de M. De acordo com nossa escolha, C_l tem q_a como símbolo de estado e pela cláusula (H) ela deve ser uma configuração de aceitação válida. Finalmente, pela cláusula (I) de ϕ_n^{M}, C_1 tem q_0 como símbolo de estado,

seguido da cadeia $w = w_1...w_n$ composta apenas por símbolos do alfabeto de entrada. Assim, $\mathcal{C}$ é um histórico de computação de aceitação de M com entrada w.

□

Mostramos no Teorema 8 que $E_{K\perp\alpha}$ é indecidível. Este resultado vem do fato que, se $E_{K\perp\alpha}$ for decidível, então o problema da vacuidade para ALL seria decidível, uma contradição.

Teorema 8. *$E_{K\perp\alpha}$ é indecidível.*

Demonstração. Prova por redução de E_{ALL}. Suponha que existe um algoritmo que R que decide $E_{K\perp\alpha}$. Construímos um algoritmo S que decide E_{ALL} como segue

S = "Sobre a entrada $\langle M\rangle$, onde M é um ALL:

1. Execute R sobre a entrada $\langle K, \alpha\rangle$, onde $K = \{\phi_i^{\mathrm{M}} \mid i \geq 0\}$ e $\alpha = \neg \mathrm{F} q_a$.
2. Se R aceita, *aceite*. Se R rejeita, *rejeite*."

Todo elemento de $K\perp\neg \mathrm{F} q_a$ é necessariamente um conjunto de fórmulas unitário, dado que para todo $\phi_i^{\mathrm{M}} \neq \phi_j^{\mathrm{M}}$, não existe um modelo LTL que satisfaça simultaneamente as cláusulas (I) e (X) em ambos ϕ_i^{M} e ϕ_j^{M}.

Se R rejeita $\langle K, \alpha\rangle$, então existe um $\{\phi_i^{\mathrm{M}}\} \in K\perp\neg \mathrm{F} q_a$ e um modelo $\mathcal{M}$ que satisfaz ϕ_i^{M} e $\mathrm{F} q_a$. Pelo Lema 7, temos que o ALL M aceita uma entrada de tamanho i, e portando a linguagem de M não é vazia.

Se R aceita $\langle K, \alpha\rangle$, então $K\perp\neg \mathrm{F} q_a = \emptyset$. Assim, para todo possível $K' = \{\phi_i^{\mathrm{M}}\}$ e cada possível modelo LTL $\mathcal{M}$, ou $\mathcal{M} \not\models \phi_i^{\mathrm{M}}$, ou $\mathcal{M} \models \neg \mathrm{F} q_a$. Dado que $K\perp\neg \mathrm{F} q_a = \emptyset$, não existe ϕ_i^{M} tal que $\phi_i^{\mathrm{M}} \wedge \mathrm{F} q_a$ é satisfazível. Pelo Lema 7 nenhuma entrada é aceita por M, assim a linguagem de M é vazia.

Se R é um decisor de $E_{K\perp\alpha}$, então S é um decisor de E_{ALL}. Contudo, a existência de S contradiz a indecidibilidade de E_{ALL}, assim deve ser o caso onde $E_{K\perp\alpha}$ é indecidível. □

Temos assim o seguinte corolário como consequência direta do Teorema 8.

Corolário. *A contração partial meet é incomputável para conjuntos de fórmulas LTL.*

Este resultado vem do fato que, para definir uma contração partial meet para LTL, precisamos verificar se o conjunto resíduo é ou não vazio. Assim, a construção dependeria de um problema indecidível.

5.3 Contração LTL Restrita

Apesar do Teorema 8 mostrar que, no caso geral, construir operadores partial meet para LTL não é factível, mostramos que sob certas restrições, ainda é possível definir corretamente este tipo de operador.

Uma possível solução para o problema seria restringir a apenas conjuntos finitos de fórmulas. Neste caso, a compacidade seguiria trivialmente e apenas monotonicidade seria necessária para assegurar a correta definição da contração partial meet [HW02]. Contudo, no caso onde a especificação do sistema é representada utilizando lógicas temporais, conjuntos infinitos de fórmulas podem aparecer mesmo em aplicações práticas relativamente simples. Considere, por exemplo, o modelo na Figura 5.5. O conjunto $\{p, Xp, XXp, ...\}$ é uma descrição intuitiva deste sistema.

Para ilustrar o problema de uma representação finita, suponha, por exemplo, que representamos o sistema descrito na Figura 2.1 por meio do seguinte conjunto finito de fórmulas $K = \{p, \mathrm{XG} p, \mathrm{XG} q\}$. O conjunto K é de fato uma representação apropriada para o modelo dado que toda fórmula LTL que o modelo satisfaz pode ser inferido de K por um operador de consequência apropriado.

Suponha agora que desejamos contrair a crença $\alpha = \mathrm{X}q$ de K. Por contração clássica, maximizando a preservação de fórmulas, resultaria em um único conjunto $K' = \{p, \mathrm{XG}p\}$. Contudo, fórmulas como $\mathrm{XX}q$ ou $p\mathrm{U}q$ deixariam de ser inferidas de K', apesar delas serem consistentes com α. Apesar de escolhas diferentes de K poderem produzir resultados adequados para essa contração, não existe representação finita que garanta a melhor escolha para todas as possíveis fórmulas α.

Mostramos a seguir que é possível definir um operador de contração partial meet com restrições mais fracas e que pode ser usado ainda para conjuntos infinitos de crenças.

5.3.1 Restrição sobre K

Nossa restrição sobre K consiste em limitar o problema para conjuntos de fórmulas que podem ser *definíveis por um modelo*.

Definição 19. Seja $\mathcal{M}$ um modelo LTL, um conjunto de crenças K é dito definível por um modelo $\mathcal{M}$ se

$$K = \{\phi \in \mathcal{L}_{LTL} \mid \mathcal{M} \models \phi\}$$

Esta representação é similar à usada por [KM91] onde o conjunto de crenças B é um conjunto infinito de fórmulas caracterizado por uma única fórmula proposicional ψ onde $B = \{\phi \mid \psi \vdash \phi\}$.

Contudo, essa restrição sobre K sozinha não é capaz de assegurar a correta definição de um operador de contração partial meet. Isso ocorre devido ao que chamamos de falha de *upperbound*.

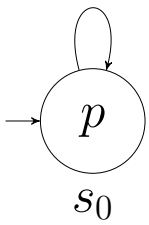


Figura 5.5: *Exemplo de modelo com um único estado.*

Proposição 9 (*falha de upperbound*)**.** *Seja $\alpha \in \mathcal{L}_{LTL}$ uma fórmula LTL, existe um conjunto K definido por um modelo tal que $\alpha \in K$, $B \subset K$ e $\alpha \notin Cn(B)$, porém $K\perp\alpha$ é vazio.*

Demonstração. Seja $\mathcal{M} = \{\{p\}, \{s_0\}, s_0, \{(s_0, s_0)\}, L(p) = \{s_0\}\}$ um modelo LTL como descrito pelo diagrama de estados na Figura 5.5, $K = \{\phi \in L_{LTL} \mid \mathcal{M} \models \phi\}$ um conjunto definido por $\mathcal{M}$ e $\alpha = \mathrm{FG}p$. Temos que $K\perp\alpha = \emptyset$ apesar de existe um número infinito de subconjuntos B de K tal que $\alpha \notin Cn(B)$.

Suponha que existe $B \in K\perp\alpha$. Como $K = Cn(\{p, \mathrm{X}p, \mathrm{XX}p, ...\})$, então, para algum $i \geq 0$, deve ser o caso que $\mathrm{X}^i p \notin B$, caso contrário $\mathrm{G}p \in Cn(B)$ e então $\mathrm{FG}p \in Cn(B)$. Deve ser o caso também que $\mathrm{X}^j p \notin Cn(B)$ para algum $j > i$, caso contrário $\mathrm{X}^{i+1}\mathrm{G}p \in Cn(B)$ e então $\mathrm{FG}p \in Cn(B)$. Contudo isso contradiz a maximalidade de B, dado que $B' = B \cup \{\mathrm{X}^i p\}$, $B \subset B' \subseteq K$ e $\alpha \notin Cn(B')$. Então, o conjunto $K\perp\alpha$ deve ser vazio. □

Devido a falha de *upperbound* é impossível definir corretamente um operador de contração *partial meet*. Neste exemplo, dado que $K\perp\alpha = \emptyset$, qualquer função de seleção γ resulta, por definição, em $\gamma(K\perp\alpha) = \{K\}$. Assim, qualquer operador partial meet definido sobre uma função de seleção γ qualquer resulta em

$$K -_\gamma \alpha = \bigcap \gamma(K\perp\alpha) = \bigcap \{K\} = K.$$

Dado que $\alpha \in K$, não existe contração partial meet capaz de satisfazer sequer o sucesso da contração esperado pela teoria AGM.

Uma vez que a restrição sobre K não é suficiente para os nossos propósitos, assumimos uma segunda restrição ao problema: uma limitação na expressividade da fórmula de entrada.

5.3.2 Restrição sobre α

Restringimos a fórmula de entrada de acordo com a propriedade temporal que ela representa. Em verificação formal, a correção de um sistema é assegurada por uma verificação sistemática de um modelo frente a um conjunto de propriedades desejadas. Lamport [Lam77] introduziu duas classes principais de propriedades: *safety* e *liveness*. Alpern e Schneider [AS85] mostram que toda propriedade linear pode ser escrita como a interseção de uma propriedade *safety* e uma propriedade *liveness*. Exploramos então a contração segundo a classe de propriedades a qual é expressada pela fórmula de entrada α.

Devemos ressaltar que a restrição sobre α, sozinha, também não é suficiente para assegurar que um operador de contração *partial meet* sempre exista. De fato, a fórmula $\alpha = \neg \mathrm{F} q_a$ no Teorema 8 é um dos tipos mais simples de propriedades temporais chamado *invariante* (Definição 20), e, contudo, o problema foi mostrado indecidível.

Investigamos a decidibilidade do problema $E_{K \perp \alpha}$, assumindo que as restrições em ambos K e α, quando K é um conjunto de fórmulas definidos por modelo e α representa uma invariante, uma propriedade *safety* ou uma propriedade *liveness*.[5]

5.3.3 Contração de invariantes

Invariantes são propriedades as quais se espera sejam verdadeiras em todo o modelo. Uma invariante descreve uma condição que deve ser satisfeita em todos os estados alcançáveis do modelo [BK08].

Em LTL, invariantes são expressadas por fórmulas $\mathrm{G}\phi$ onde ϕ é uma fórmula da lógica proposicional.

Definição 20. Seja $\mathcal{M}$ um modelo LTL, uma fórmula proposicional ψ expressa uma propriedade invariante em $\mathcal{M}$ se $\mathcal{M} \models \mathrm{G}\psi$.

Para abordar o problema da vacuidade para invariantes, usamos o conceito de *desdobramento* de modelos. Primeiro, definimos como representar o que chamamos de codificação de *frame*.

Definição 21. Seja $\mathcal{M} = \langle AP, S, s_0, R, L \rangle$ um modelo LTL, a codificação de *frame* $\lceil \mathcal{M} \rceil$ de $\mathcal{M}$ é uma cadeiauv^+ tal que

1. $u = s_0 s_1 ... s_{m-1}$ e $v = s_m s_{m+1} ... s_n$, e
2. para todo $i < n$, $(s_i, s_{i+1}) \in R$, e $(s_n, s_m) \in R$

O desdobramento de um modelo $\mathcal{M}$ é um modelo bissimilar $\mathcal{M}'$ tal que, para $\lceil \mathcal{M} \rceil = uv^+$, a codificação do *frame* $\mathcal{M}'$ é $\lceil \mathcal{M}' \rceil = uvv^+$. Isso representa intuitivamente uma "extração" de estados do laço de $\mathcal{M}$ replicando-os para a porção sem repetição.

Definição 22. Seja $\mathcal{M}$ um modelo LTL e $\lceil \mathcal{M} \rceil = uv^+$ sua codificação de *frame*, um modelo $\mathcal{M}_k$ é um *k-desdobramento* de $\mathcal{M}$ se

1. $\lceil \mathcal{M}_k \rceil = zv^+$, onde $|z| = |uv^k|$, e
2. $\mathcal{M}$ e $\mathcal{M}_k$ são bissimilares.

O Teorema 10 mostra que, para um conjunto de crenças K definido por um modelo e por uma fórmula LTL $\alpha = \mathrm{G}\phi$ representando uma invariante, se ϕ não é uma tautologia, o conjunto resíduo $K \perp \alpha$ é não vazio.

Teorema 10. *Seja $\mathcal{M}$ um modelo LTL, K um conjunto de crenças K definido por $\mathcal{M}$ e $\alpha = \mathrm{G}\psi$ uma fórmula LTL, onde ψ é uma invariante em $\mathcal{M}$,*

[5] Apesar de invariantes serem um tipo especial de propriedades *safety*, escolhemos aborda-la separadamente dado sua importância em aplicações de verificação formal.

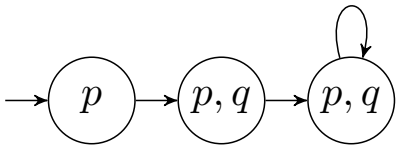


Figura 5.6: *O 1-desdobramento de $\mathcal{M}$ na Figura 2.1.*

se $\not\models \psi$, então $K\bot\alpha \neq \emptyset$.

Ideia da prova. Definimos um modelo $\mathcal{M}'$ tal que $\mathcal{M}'$ tem a mesma codificação de *frame* do 1-desdobramento de $\mathcal{M}$. Definimos então a função de rotulação de $\mathcal{M}'$ igual a de $\mathcal{M}$ exceto pelas proposições relacionadas ao estado inicial, ao qual é minimamente modificada de modo a fazer $\neg\psi$ satisfeito neste estado (revisão de crenças clássica pode ser realizada para esse objetivo). Mostramos então que $K' = \{\phi \in K \mid \mathcal{M}' \models \phi\}$ pertence a $K\bot\alpha$ por uma prova direta que K' é um subconjunto maximal de K que não implica α. E assim, $K\bot\alpha \neq \emptyset$. □

Demonstração. Seja AP_α o conjunto de átomos proposicionais de α e $\mathcal{M}_1$ um 1-desdobramento de $\mathcal{M}$. Definimos $\mathcal{M}'$ a partir de $\mathcal{M}_1$ tal que $S' = S$, $R' = R$ e, para $B = \{l \mid l{=}p$ ou $l{=}\neg p$, $p \in P_\alpha$ e $\mathcal{M} \models l\}$ e B' um elemento de $B\bot\phi$, a função de rotulação L' é definida como

$$L'(p) = \begin{cases} L(p) \setminus \{s_0\}, & \text{se } p \in B \text{ e } p \notin B', \\ L(p) \cup \{s_0\}, & \text{se } \neg p \in B \text{ e } \neg p \notin B', \\ L(p), & \text{caso contrário.} \end{cases}$$

A base $K' = \{\phi \in K \mid \mathcal{M}' \models \phi\}$ pertence a $K\bot\alpha$. Por construção, $K' \subseteq K$ e $\phi \notin Cn(K')$, então $\alpha \notin Cn(K')$. Resta mostrar que não existe fórmula $\beta \in K$ tal que $\beta \notin K'$ e $\alpha \notin Cn(K' \cup \{\beta\})$.

Se $\beta = p$ ou $\beta = \neg p$, dado que $B' \in B\bot\phi$, vale que $\phi \in Cn(B' \cup \{\beta\})$. Como $Cn(B') \subseteq Cn(K')$, também vale que $\phi \in Cn(K' \cup \{\beta\})$. Por construção, $\mathrm{X}\phi \in K'$, assim $\phi \wedge \mathrm{X}\phi \in Cn(K' \cup \{\beta\})$ e $\alpha = \mathrm{G}\phi \in Cn(K' \cup \{\beta\})$.

Como hipótese de indução, assumamos que para as fórmulas β_1 e β_2 vale que $\phi \in Cn(K' \cup \{\beta_1\})$ e $\phi \in Cn(K' \cup \{\beta_2\})$.

Se $\beta = \beta_1 \mathrm{R} \beta_2$, todo modelo de $\beta_1 \mathrm{R} \beta_2$ satisfaz β_2. Dado que $\beta_2 \in Cn(\beta_1 \mathrm{R} \beta_2)$ e, por hipótese de indução, $\phi \in Cn(K' \cup \{\beta_2\})$, vale que $\phi \in Cn(K' \cup \{\beta\})$ e assim $\alpha = \mathrm{G}\phi \in Cn(K' \cup \{\beta\})$.

Se $\beta = \beta_1 \mathrm{U} \beta_2$, cada modelo $\mathcal{M}''$ de $\beta_1 \mathrm{U} \beta_2$ satisfaz β_2 or $\beta_1 \wedge \mathrm{X}(\beta_1 \mathrm{U} \beta_2)$. Por hipótese de indução, $\mathcal{M}''$ é um modelo de ϕ, assim vale que $\phi \in Cn(K' \cup \{\beta\})$ e então $\alpha = \mathrm{G}\phi \in Cn(K' \cup \{\beta\})$.

Portanto, para toda fórmula $\beta \in K$, se $\beta \notin K'$, então $\alpha \in Cn(K' \cup \{\beta\})$. Assim, não existe conjunto K'' tal que $K' \subset K'' \subseteq K$ e $\alpha \notin Cn(K'')$, portanto $K' \in K\bot\alpha$. □

5.3.4 Contração de propriedades *safety*

Propriedades *safety* são generalizações de invariantes onde podemos descrever o comportamento esperado de uma sequência de eventos, em vez de analisá-lo independentemente em cada estado. Propriedades típicas dessa classe são exclusão mutual e *deadlock freedom*. De modo informal, propriedades *safety* expressam que "algo (ruim) não acontece" [Lam77]. Abordamos então o problema da vacuidade de $K\bot\alpha$ quando a fórmula de entrada representa uma propriedade *safety*.

A definição formal de propriedades *safety* é dada em termos de contraexemplos. Toda computação que refuta uma propriedade *safety* tem um prefixo finito que testemunha a falha da propriedade [AS85].

Definição 23. Dizemos que π é um *contraexemplo finito* para α em $\mathcal{M}$ se π é um caminho finito $s_0 \to s_1 \to ... \to s_n$ em $\mathcal{M}$ tal que $\pi \models \neg\alpha$.

Definição 24. Uma fórmula LTL α expressa uma propriedade *safety* se e somente se para todo modelo LTL $\mathcal{M}$, se $\mathcal{M} \not\models \alpha$, então existe um contraexemplo finito para α em $\mathcal{M}$.

Essa definição estipula que se uma violação de uma propriedade *safety* ocorre, existe um ponto da execução que tal violação pode ser reconhecida[AS85].

De modo a definir o conjunto resíduo, fazemos uso do que chamamos de *supressão de* α: mudanças mínimas no modelo de modo a introduzir contraexemplos para α.

Definição 25. Seja $\mathcal{M}$ um modelo LTL, um modelo $\mathcal{M}'$ é uma supressão de α de $\mathcal{M}$ admissível se

1. $\lceil \mathcal{M}' \rceil = \lceil \mathcal{M} \rceil = uv^+$
2. $\mathcal{M}' \not\models \alpha$
3. Para todo $p \in AP$
   (a) $L'(p) = L(p)$ se p não é uma subfórmula de α,
   (b) para $v = v_0 v_1 ... v_n$, se $v_i \in L(p)$, então $v_i \in L'(p)$.

Essa definição significa intuitivamente que $\mathcal{M}'$ é uma supressão de α de $\mathcal{M}$ admissível se ele pode ser construído a partir de $\mathcal{M}$ mudando a função de rotulação apenas nos estados de $\mathcal{M}$ que não estão fora da porção do laço.

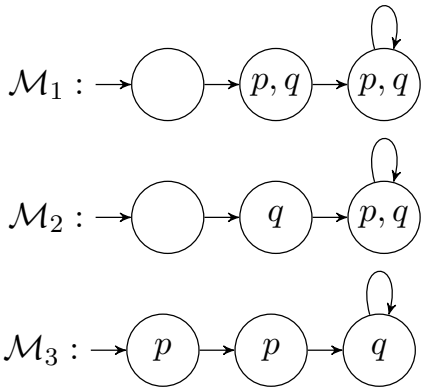


Figura 5.7: *Modificação no modelo da Figura 5.6.*

Na Figura 5.7, $\mathcal{M}_1$ e $\mathcal{M}_2$ são exemplos de supressões de Gp no modelo da Figura 5.6. O modelo $\mathcal{M}_3$ não é uma modificação admissível dado que viola ambas as condições 3(a) e 3(b) da definição.

A minimalidade de uma modificação que suprime α é medida sobre conjuntos de fórmulas chamadas *elementares*.

Definição 26. Seja K um conjunto de crenças definidos por um modelo, denotamos por E_K o conjunto de fórmulas elementares de K onde

$$\mathrm{E}_K = \{\mathrm{X}^i\phi \in K \mid \phi = p \text{ ou } \phi = \neg p, p \in AP \text{ e } i \geq 0\}$$

Dizemos que $\mathcal{M}'$ é uma supressão de α em $\mathcal{M}$ minimal se $\mathcal{M}' \not\models \alpha$ e maximiza a preservação de fórmulas elementares satisfeitas por $\mathcal{M}$.

Definição 27. Um modelo $\mathcal{M}'$ é uma supressão de α em $\mathcal{M}$ minimal se

1. $\mathcal{M}'$ é admissível
2. não existe outra supressão de α em $\mathcal{M}$ admissível, $\mathcal{M}''$, tal que $\{\phi \in \mathrm{E}_K \mid \mathcal{M}' \models \phi\} \subset \{\phi \in \mathrm{E}_K \mid \mathcal{M}'' \models \phi\}$

Mostramos no Teorema que, para um conjunto de crenças K definidos por um modelo e uma fórmula α que representa uma propriedade *safety*, se α não é uma fórmula válida, o conjunto resíduo $K \perp \alpha$ é não vazio.

Teorema 11. *Seja $\mathcal{M}$ um modelo LTL, K um conjunto de crenças definidos por um modelo $\mathcal{M}$ e α uma fórmula LTL expressando uma propriedade safety,*

$$\text{se } \not\models \alpha, \text{ então } K\bot\alpha \neq \emptyset.$$

Proof Sketch.. Seja $\mathcal{M}_k$ o menor k-desdobramento de $\mathcal{M}$ tal que podemos construir uma supressão de α em $\mathcal{M}$ admissível. Dado que α é uma propriedade *safety*, sempre existirá tal k-desdobramento onde podemos modificar apenas sua sequência inicial de estados de modo a introduzir um dos contraexemplos finitos para α. Assim, tal modelo $\mathcal{M}_k$ sempre existe. Seja $\mathcal{M}'$ a supressão de α minimal, mostramos que $K' = \{\phi \in K \mid \mathcal{M}' \models \phi\}$ pertence a $K\bot\alpha$ por uma prova de que K' é um subconjunto maximal de K que não implica α. E assim, vale que $K\bot\alpha \neq \emptyset$. □

5.3.5 Falha da contração de propriedades *liveness*

Diferente de propriedades *safety*, propriedades *liveness* são usados para estipular que "boas coisas" devem acontecer durante a execução de um sistema [Lam77]. A propriedade de *starvation-freedom* é um exemplo típico de propriedade *liveness*.

Uma característica importante de propriedades *liveness* é que elas só são violadas por caminhos de computação infinitos [AS85], isto é, não existe um ponto no caminho de execução onde a violação de uma propriedade *liveness* possa ser identificada olhando apenas para os eventos passados.

Contudo, o molde de restrições usado até então não é suficiente para uma definição adequada um operador de contração partial meet para propriedades do tipo *liveness*. Na Proposição 9, a fórmula FGp é de fato uma propriedade *liveness* : ela requer que em algum ponto de uma computação, p globalmente seja válido. Como visto, devido a falha de *upperbound*, não podemos definir corretamente o operador de contração partial meet, mesmo neste caso onde o conjunto de crenças é definível por um modelo.

Alguns autores, todavia, argumentam que para propósitos práticos de verificação, apenas propriedades *safety* são úteis [BBF+01]. Eles argumentam que propriedades *liveness* apenas asseguram que um evento irá ocorrer, sem, contudo, fornecer informações sobre tempo envolvido. Do ponto de vista utilitário, dado que não podemos esperar indefinidamente, as propriedades *liveness* trazem pouca informação para a verificação[BBF+01].

Uma solução seria fortalecer propriedades *liveness* com limitações no tempo, contudo estaríamos de fato definindo um tipo de propriedades *safety*. Neste sentido, apesar de não conseguirmos definir contração partial meet para propriedades do tipo *liveness*, nossos resultados para *safety* são úteis para a maioria dos casos práticos.

5.4 Indecidibilidade de *partial meet* para outras lógicas

A indecidibilidade de $E_{K\bot\alpha}$ ocorre também para outras lógicas temporais e modais, como CTL [CE82] e PDL [FL79, HKT00].

Para CTL, o resultado pode ser mostrado com as mesmas demonstrações apresentadas utilizadas para LTL, com mudanças em algumas definições para fórmulas correspondentes em CTL. Por exemplo, uma fórmula ALL-equivalente em CTL é dada pela mesma construção da Definição 18, com a substituição dos operadores temporais X e G por AX e AG, respectivamente.

Com isso, podemos demonstrar que um ALL M aceita w se e somente se a fórmula $\phi^{\mathrm{M}}_{|w|} \wedge \mathrm{EF}q_a$ é satisfeita. Esta propriedade é então usada como no Teorema 8, com a substituição de $\neg\mathrm{F}q_a$ por $\neg\mathrm{EF}q_a$, de modo a mostrar que $E_{K\bot\alpha}$ também é indecidível para CTL.

Similarmente, podemos mostrar que $E_{K\bot\alpha}$ é indecidível para PDL. Neste caso, a construção da fórmula ALL-equivalente é feita pela substituição dos operadores temporais X e G por $[\pi]$ e $[\pi^*]$, respectivamente, onde π é um programa atômico arbitrário.

Podemos mostrar que um ALL M aceita w se e somente se a fórmula PDL $\phi^{\mathrm{M}}_{|w|} \wedge \langle\pi^*\rangle q_a$ é satisfazível. Como antes, esta propriedade pode ser usada para produzir uma prova similar àquela do Teorema 8, substituindo $\neg\mathrm{F}q_a$ por $\neg\langle\pi^*\rangle q_a$, a fim de mostrar que $E_{K\bot\alpha}$ também é indecidível para PDL.

5.5 Notas finais sobre o capítulo

O problema da vacuidade de conjuntos resíduos é indecidível, dado que podemos reduzir a este o problema da vacuidade de linguagem de autômatos linearmente limitados. Este resultado implica que no caso geral, é impossível definir um operador de contração partial meet para conjuntos de fórmulas LTL, como definido por AGM.

Contudo, mostramos que ainda é possível abordar o problema limitando os conjuntos de fórmulas LTL para aqueles definidos por modelos e restringindo a fórmula de entrada para fórmulas que expressem propriedades *safety*, evitando deste modo a o problema da indecidibilidade.

O trabalho de van Zee et al.[VDDV15] apresenta uma lógica para revisão de bases de crenças temporais, junto com uma construção para a revisão baseada em minimalidade de modelos. Podemos ver que eles evitam o problema da indecidibilidade por considerar crenças apenas até certo ponto no tempo. Uma ideia similar foi proposta por Finger e Wassermann [FW08], onde em vez de realizar verificação de modelos tradicional, uma versão com limitação no tempo foi utilizada. Isto corresponde a limitar o futuro e verificar modelo até este ponto no tempo. Estes resultados apoiam nossa afirmação que, apesar do resultado da indecidibilidade, a teoria AGM ainda pode ser aplicada com sucesso em diversos domínios que envolvem logicas temporais, dado que utilizemos as restrições adequadas.

Capítulo 6

Reparo estrutural de modelos

Neste capítulo especificamos nossa abordagem para mudança de crenças CTL, a qual denominamos *revisão de modelos CTL*. Nossa abordagem baseia-se em revisão de crenças e difere da atualização de modelos de Zhang e Ding [ZD08] da mesma maneira que diferem revisão e atualização de crenças: o contexto ao qual é aplicada. Ilustramos essa diferença e argumentamos o porquê de não usar diretamente a teoria clássica. Definiremos formalmente nosso operador de revisão e analisaremos sua obediência aos postulados AGM.[1]

Na Seção 6.1 mostramos um caso onde a atualização de modelos não retorna o resultado esperado. Na Seção 6.2 apresentamos um trabalho relacionado a nossa abordagem de reparo estrutural de modelos. Por último, definimos na Seção 6.3 nosso operador de revisão de modelos CTL e mostramos seu alinhamento com os postulados AGM.

6.1 Atualização Modelos em um Contexto Estático

A necessidade de uma nova abordagem (para mudanças em crenças CTL) é motivada pela diferença fundamental entre revisão e atualização de crenças. A abordagem de [ZD08] deve ser aplicada quando desejamos incorporar à nossas crenças uma informação que retrata uma mudança no mundo. Desejamos deste modo adaptar nossa base de crenças a nova configuração que o mundo assumiu. Por outro lado, aplicamos revisão de crenças quando lidamos com um contexto estático, onde a nova informação nos permite refinar (ou corrigir) nossas crenças sobre um mundo que não mudou. O Exemplo 6.1.1 ilustra essa diferença.

Exemplo 6.1.1. *Suponha que em um quarto existem três objetos: um livro, uma revista e uma mesa. Acreditamos que, ou o livro está sobre a mesa (b) ou a revista está (m), mas não ambos. Suponhaque enviamos o robô apenas para observar a posição do livro, sem alterá-la. Suponha então que quando o robô sai da sala ele nos informa que o livro está sobre a mesa. Desejamos assim incorporar a nova informação* $\phi \leftrightarrow b$ *à nossa base de crenças* $\psi \leftrightarrow (b \wedge \neg m) \vee (\neg b \wedge m)$.

Um operador de atualização geraria a nova base de crença $\psi' \leftrightarrow b$*, enquanto um operador de revisão geraria uma base* $\psi'' \leftrightarrow (b \wedge \neg m)$*. Observe que, costumávamos acreditar que o livro e a revista não estavam no mesmo lugar, fomos informados sobre a posição do livro e sabemos que a sala não foi alterada, assim não temos motivos para duvidar que a posição da revista ainda seja diferente da posição do livro. Assim, a revisão de crenças produziu um resultado mais informativo que a atualização.*

Observe que o que muda entre revisão de crenças e atualização de crenças é como o robô interage com o mundo e como interpretamos isso. Acreditamos que o mesmo problema de distinção entre mundos estáticos e dinâmicos acontece também quando lidamos com o reparo de modelos inconsistentes da lógica CTL.

[1] Uma versão preliminar dessa abordagem foi apresentada na 12a. Conferência Ibero-americana de Inteligência Artificial, Bahía Blanca, Argentina, 2010.

Existem dois principais motivos para modificar a modelagem de um sistema: (1) quando ela falha em satisfazer algum requisito da especificação formal e (2) quando ela precisa ser adaptada a um novo requisito. Quando modificamos uma especificação motivado por (1), repará-la por meio de uma abordagem baseada em revisão de crenças pode gerar um resultado mais informativo que aquele gerado pela atualização de modelos CTL, de um modo similar ao que ocorreu entre revisão e atualização de crenças no Exemplo 6.1.1.

No Exemplo 6.1.2 ilustramos uma situação onde um modelo de sistema falha em atender um requisito desejado. Veremos que neste exemplo a atualização de modelos gera possibilidades de solução que fogem à intenção inicial do sistema.

Exemplo 6.1.2. *Suponha que projetamos um computador alimentado por uma fonte infinita de energia. Nossa intenção é oferecer um equipamento para um uso ininterrupto, porém tivemos que incluir duas restrições no projeto: o computador deve, por razões de segurança, permanecer desligado até chegar ao consumidor final; e o computador deve eventualmente realizar algumas rotinas de atualização de software.*

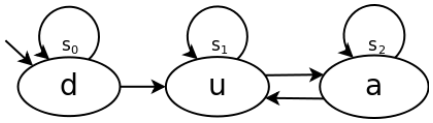


Figura 6.1: *Modelagem que satisfaz a especificação ψ.*

Seja d, u e a proposições que representem, respectivamente, os estados "desligado", "em uso" e "atualizando", a especificação do nosso sistema é dada pela seguinte fórmula CTL

$$\begin{aligned}\psi \quad \leftrightarrow \quad & d \ \wedge \ AG(d \rightarrow AX(d \vee u)) \ \wedge \ AG(u \rightarrow AX(u \vee a)) \ \wedge \ AG(a \rightarrow AX(u \vee a)) \\ & \wedge \ AG((d \wedge \neg u \wedge \neg a) \vee (\neg d \wedge u \wedge \neg a) \vee (\neg d \wedge \neg u \wedge a))\end{aligned}$$

A Figura 6.1 representa um possível modelo para o sistema. Depois de um tempo, alguns usuários queixaram-se que não estavam conseguindo utilizar o computador porque ele estava em constante atualização. Analisando a queixa, verificamos então que nosso projeto não garante que o estado "em uso" sempre possa ser alcançado. Em outras palavras, descobrimos que existem modelos em $Mod(\psi)$ que não satisfazem a seguinte fórmula CTL

$$\phi \quad \leftrightarrow \quad EF(u) \ \wedge \ AF(u)$$

A Figura 6.2 representa alguns dos modelos de ψ que não são modelos de ϕ. Acreditávamos que todos os modelos de ψ eram adequados, agora desejamos identificar um novo conjunto de modelos que satisfaçam ϕ e que sejam próximos à especificação original. Para isso, faremos uso de atualização de modelos e de seu operador $\diamond_c$.

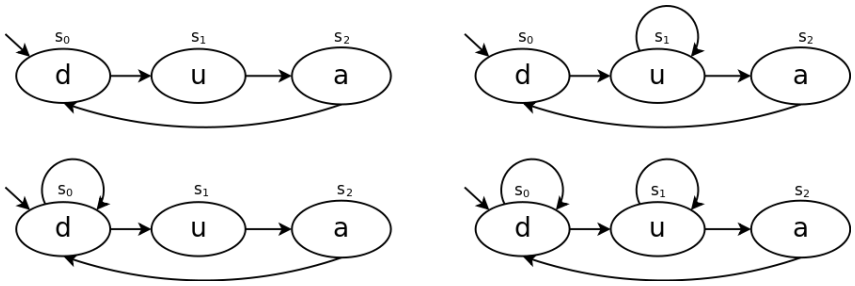


Figura 6.2: *Exemplos de modelos de ψ que não satisfazem ϕ.*

Pela definição de $\diamond_c$, o operador buscará os modelos de ϕ mais próximos aos modelos de ψ, analisando cada um destes individualmente. Tomemos os modelos M, M_1 e M_2 descritos na Figura 6.3. O modelo M pertence a $Mod(\psi)$, logo todo $M' \in Mod(\phi)$ minimal segundo $\leqslant_M$ pertencerá ao conjunto $Mod(\psi \diamond_c \phi)$. Dessa forma, temos que M_1 e M_2 pertencem ao conjunto $Mod(\psi \diamond_c \phi)$.

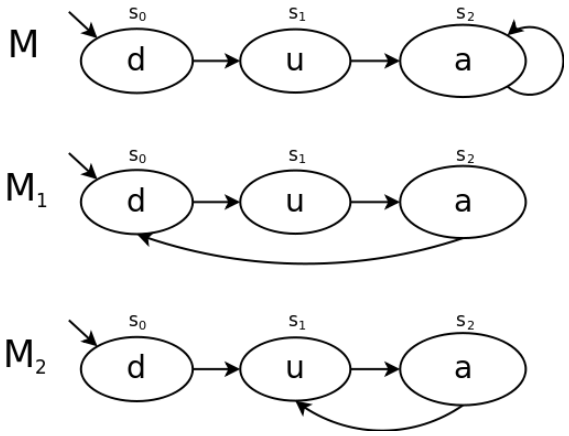


Figura 6.3: *Exemplos de possíveis atualizações de um modelo.*

Além de M_1, outros modelos de $\psi \diamond_c \phi$ possuem a transição de um estado "atualizando" para um estado "desligado". Como podemos ver, a atualização de modelos cogita como solução o desligamento do computador para que ele possa alcançar novamente o estado de uso. Porém isso não nos é intuitivo: por que desligar um computador que pode ficar permanentemente ligado?

O problema de utilizar atualização de modelos no Exemplo 6.1.2 diz respeito a natureza do novo requisito. O requisito era uma propriedade que o projeto já deveria satisfazer e não uma mudança a qual ele deveria se adequar. Percebemos que nossa especificação era indiferente ao requisito (alguns modelos satisfaziam, outros não) e isso estava incorreto. Porém, o natural seria manter apenas os modelos que já satisfazem a especificação e o requisito, e não adicionar novas possibilidades, como obtemos usando o operador $\diamond_c$. Mais à frente neste capítulo, veremos que uma abordagem baseada em revisão de crenças pode nos fornecer o conjunto de modelos que desejamos.

6.2 Primeiras abordagens: Belief Revision NuSMV

A integração dos conceitos de verificação de modelos e revisão de crenças é abordada nos trabalhos [Sou07, SW07], com a criação do BrNuSMV[2]: um verificador capaz de gerar sugestões de atualização quando um dado modelo se mostra inconsistente com sua especificação. Resultado de um projeto de mestrado, o BrNuSMV consiste no verificador NuSMV [CCGR99] integrado de uma camada que, por meio do uso de técnicas de revisão de crença, é capaz de gerar sugestões de mudança no modelo de entrada. O trabalho propõe um algoritmo de revisão baseado no sistema de esferas para tratar caminhos diagnosticados como inconsistentes pelo verificador de modelos.

O algoritmo inicia verificando a consistência entre a descrição de um sistema e sua especificação. Quando inconsistente, o caminho contraexemplo é identificado e então se inicia o processo de revisão.

Buscando no caminho contraexemplo o último estado consistente, assume-se como crença as proposições válidas neste estado. Cria-se então um sistema de esfera centralizado nos mundos onde tal crença é válida. Este sistema de esferas é ordenado por uma função d, que indica quantas proposições diferentes um mundo possui da crença (mundos com a mesma "distância" da crença pertencem a mesma esfera). Por fim, um operador de revisão para um sistema de esferas é aplicado, resultando em um pequeno conjunto de mundos consistentes com a especificação.

A última etapa é eliminar a alcançabilidade da inconsistência (Figura 6.4). Há duas possibilidades: eliminando a transição que leva ao estado inconsistente ou adicionando uma transição que se sobreponha a transição inconsistente, direcionando o fluxo a um estado do conjunto resultante da revisão. Essas sugestões são reunidas e retornadas como resultado. Maiores detalhes sobre o algoritmo podem ser vistos em [Sou07].

[2] Acrônimo para *Belief Revision NuSMV*.

Figura 6.4: *Eliminando da inconsistência no BrNuSMV [Sou07].*

Essa abordagem explorou o uso prático de revisão de crenças e trouxe importantes contribuições para a área, sendo uma das primeiras implementações de revisão em uma ferramenta utilizada em modelagem formal. O trabalho mostra a eficácia da ferramenta, e consequentemente da revisão de crenças, como método de auxílio ao desenvolvimento de sistemas computacionais.

Talvez por ser um passo inicial na integração de verificação de modelos e revisão de crenças, existem no BrNuSMV algumas lacunas: não é possível tratar todos os tipos de fórmulas CTL; o processo de revisão é feito estado a estado, com limitações nos conjuntos de crenças; não há uma análise formal do processo de revisão de modelos. No nosso trabalho preenchemos algumas dessas lacunas. Realizamos uma análise formal do processo de revisão de modelos CTL genéricos e estudamos sua relação com abordagens clássicas de revisão de crenças.

6.3 Operador de Revisão de Modelos

Antes de especificar nosso método de revisão de modelos, precisamos definir uma relação de ordem entre modelos baseada na distância que um modelo possui de um dado conjunto de modelos.

Definição 28 (Ordenação por proximidade a conjuntos)**.** Seja $\mathcal{W}$ um conjunto de modelos CTL e $M_1 = (S_1, R_1, L_1)$ e $M_2 = (S_2, R_2, L_2)$ dois modelos CTL quaisquer, dizemos que M_1 é pelo menos tão próximo a $\mathcal{W}$ quanto M_2, denotado por $M_1 \leqslant_{\mathcal{W}} M_2$, se e somente se para cada conjunto de operações básicas PU1-PU5 que transforme um modelo $M' \in \mathcal{W}$ em M_2, existe um conjunto de operações básicas PU1-PU5 que transforme um modelo $M \in \mathcal{W}$ em M_1 tal que:

1. Para cada i $(1 \leq i \leq 5)$, $Diff_{PUi}(M, M_1) \subseteq Diff_{PUi}(M', M_2)$
2. Se $Diff_{PU3}(M, M_1) = Diff_{PU3}(M', M_2)$, então $\forall s \in Diff_{PU3}(M, M_1)$, $diff(L(s), L_1(s)) \subseteq diff(L'(s), L_2(s))$, onde $diff(A, B) = (A - B) \cup (B - A)$ para quaisquer conjuntos A e B.

Denotamos por $M_1 <_{\mathcal{W}} M_2$ se $M_1 \leqslant_{\mathcal{W}} M_2$ e $M_2 \not\leqslant_{\mathcal{W}} M_1$.

Como ilustração, considere novamente o Exemplo 6.1.2. Sejam M_1 e M_2 os modelos da Figura 6.3 e $\mathcal{W} = Mod(\psi)$, sabemos que M_2 pertence a $\mathcal{W}$, mas M_1 não. Assim, existe um modelo $M \in \mathcal{W}$ $(M = M_2)$ tal que para todo i $(1 \leq i \leq 5)$, $Diff_{PUi}(M, M_2) = \emptyset$. Temos que $Diff_{PUi}(M, M_2) \subseteq Diff_{PUi}(M', M_1)$, para qualquer i $(1 \leq i \leq 5)$ e $M' \in \mathcal{W}$, e portanto que $M_2 \leqslant_{\mathcal{W}} M_1$. Por outro lado, como $M_1 \notin \mathcal{W}$, qualquer que seja $M' \in \mathcal{W}$, ao menos uma das operações PUi deve ser aplicada para transformar M' em M_1. Em outras palavras, existe i $(1 \leq i \leq 5)$, tal que $Diff_{PUi}(M', M_1) \neq \emptyset$, e assim que $Diff_{PUi}(M', M_1) \not\subseteq Diff_{PUi}(M, M_2)$. Temos então $M_1 \not\leqslant_{\mathcal{W}} M_2$ e, portanto, $M_2 <_{\mathcal{W}} M_1$.

Com base na Definição 28, especificamos a seguir nosso operador de revisão $\circ_c$ para fórmulas da lógica temporal CTL.

Definição 29. Dadas duas fórmulas CTL ψ e ϕ, denotamos por $\psi \circ_c \phi$ uma fórmula CTL resultado da atualização de ψ por ϕ, tal que

$$Mod(\psi \circ_c \phi) = Min_{Mod(\psi)}(Mod(\phi))$$

onde $Min_{\mathcal{B}}(\mathcal{A})$ denota o conjunto de todos os modelos minimais de $\mathcal{A}$ com relação a ordem $\leqslant_{\mathcal{B}}$.

Para entender a seleção realizada por $\circ_c$, voltemos ao Exemplo 6.1.2. Em vez de usar o operador $\diamond_c$ para compor o novo conjunto de modelos, utilizaremos o nosso operador de revisão. Segundo a Definição 29, $\circ_c$ selecionará os modelos de ϕ que sejam minimais segundo a ordem $\leqslant_{Mod(\psi)}$, isto é, $Mod(\psi \circ_c \phi)$ conterá apenas os modelos de ϕ que mais se aproximem do conjunto de modelos de ψ. Seja M_1 e M_2 os modelos descritos pela Figura 6.3, vimos que $M_2 <_{Mod(\psi)} M_1$, logo M_1 não é minimal segundo a ordem $<_{Mod(\psi)}$ e portanto $M_1 \notin Mod(\psi \circ_c \phi)$. De fato, como $Mod(\psi) \cap Mod(\phi) \neq \emptyset$, nenhum modelo $M \notin Mod(\psi)$ pertencerá a $Mod(\psi \circ_c \phi)$, pois sempre seremos capazes de escolher um $M' \in Mod(\phi)$ tal que $M' <_{Mod(\psi)} M$. Isso significa que não cogitaremos, por exemplo, desligar o computador como solução para o problema, já que isso não condiz com nossa especificação original. O operador de revisão manterá a coerência com a especificação, selecionando apenas os modelos que já satisfazem esta e o novo requisito, sem a adição de outras possibilidades.

6.3.1 Propriedades Semânticas

O Teorema 12 descreve o principal relacionamento da nossa abordagem com a teoria clássica de revisão de crenças: a obediência aos postulados AGM.

Teorema 12. *O operador $\circ_c$ satisfaz os postulados de revisão (R1)-(R6).*

Demonstração. O postulado (R1) é satisfeito diretamente pela definição de $\circ_c$. Para satisfazer (R2) temos que provar que, para $\psi \wedge \phi$ satisfazível, $Mod(\psi \circ_c \phi) = Mod(\psi \wedge \phi)$. Se $\psi \wedge \phi$ é satisfazível, então $Mod(\psi \wedge \phi) = Mod(\psi) \cap Mod(\phi) \neq \emptyset$. Todo $M \in Mod(\psi) \cap Mod(\phi)$ é modelo de ϕ e tem distância mínima do conjunto $Mod(\psi)$ por já pertencer ao conjunto, logo $Mod(\psi \wedge \phi) \subseteq Mod(\psi \circ_c \phi)$. Suponha agora que $Mod(\psi \circ_c \phi) \not\subseteq Mod(\psi \wedge \phi)$, assim deve existir ao menos um modelo $M' \in Mod(\psi \circ_c \phi)$ tal que $M' \notin Mod(\psi) \cap Mod(\phi)$. Como, por definição, escolhemos sempre modelos de $Mod(\phi)$, necessariamente $M' \notin Mod(\psi)$. Porém, qualquer que seja $M'' \in Mod(\psi) \cap Mod(\phi)$, temos $M'' <_{Mod(\psi)} M'$, o que contradiz o fato de M' ser minimal segundo a ordem $\leqslant_{Mod(\psi)}$. Assim, $Mod(\psi \wedge \phi) \subseteq Mod(\psi \circ_c \phi)$ e portanto $Mod(\psi \wedge \phi) = Mod(\psi \circ_c \phi)$.

Agora provamos que $\circ_c$ satisfaz (R3). Se ϕ é satisfazível, então $Mod(\phi) \neq \emptyset$, o que significa que existem modelos a serem selecionados por $\circ_c$, logo $Mod(\psi \circ_c \phi) \neq \emptyset$ e, portanto, $\psi \circ_c \phi$ é satisfazível.

Para provar que $\circ_c$ satisfaz (R4) é suficiente mostrar que se $\psi_1 \leftrightarrow \psi_2$ e $\phi_1 \leftrightarrow \phi_2$, então $Mod(\psi_1 \circ_c \phi_1) = Mod(\psi_2 \circ_c \phi_2)$. Assumindo $Mod(\psi_1) = Mod(\psi_2)$ e $Mod(\phi_1) = Mod(\phi_2)$, temos que $Mod(\psi_1 \circ_c \phi_1) = Min_{Mod(\psi_1)}(Mod(\phi_1)) = Min_{Mod(\psi_1)}(Mod(\phi_2)) = Min_{Mod(\psi_2)}(Mod(\phi_2)) = Mod(\psi_2 \circ_c \phi_2)$.

Para provar que $\circ_c$ satisfaz (R5), devemos mostrar que $Mod((\psi \circ_c \phi) \wedge \mu) \subseteq Mod(\psi \circ_c (\phi \wedge \mu))$. Se $(\psi \circ_c \phi) \wedge \mu$ não é satisfazível, então $Mod((\psi \circ_c \phi) \wedge \mu) = \emptyset$ e a demonstração é trivial. Consideremos então apenas o caso onde $(\psi \circ_c \phi) \wedge \mu$ é satisfazível. Suponhamos que $Mod((\psi \circ_c \phi) \wedge \mu) \subseteq Mod(\psi \circ (\phi \wedge \mu))$ não seja verdade. Isso quer dizer que existe $M \in Mod((\psi \circ_c \phi)) \cap Mod(\mu)$ tal que $M \notin Mod(\psi \circ (\phi \wedge \mu))$. Por (R1), sabemos que $M \in Mod(\phi)$ e portanto que $M \in Mod(\phi \wedge \mu)$. Para que $M \notin Mod(\psi \circ (\phi \wedge \mu))$ deve existir um modelo $M' \in Mod(\psi \circ (\phi \wedge \mu))$ tal que $M' <_{Mod(\psi)} M$. Porém, $M' \in Mod(\phi)$, o que contradiz o fato de M ser minimal segundo $\leqslant_{Mod(\psi)}$. Logo, $Mod((\psi \circ_c \phi) \wedge \mu) \subseteq Mod(\psi \circ (\phi \wedge \mu))$ é valido.

Finalmente, mostramos que $\circ_c$ satisfaz (R6). Para tanto devemos provar que $Mod(\psi \circ_c (\phi \wedge \mu)) \subseteq Mod((\psi \circ_c \phi) \wedge \mu)$, quando $Mod((\psi \circ_c \phi) \wedge \mu) \neq \emptyset$. Se $\psi \circ_c (\phi \wedge \mu)$ não é satisfazível, a demonstração é trivial. Consideremos então apenas o caso onde esta fórmula é satisfazível. Suponha então que $Mod(\psi \circ_c (\phi \wedge \mu)) \not\subseteq Mod((\psi \circ_c \phi) \wedge \mu)$, isso significa que existe $M \in Mod(\psi \circ_c (\phi \wedge \mu))$ tal que $M \notin Mod(\psi \circ_c \phi) \cap Mod(\mu)$. Por (R1), sabemos que $M \in Mod(\phi \wedge \mu)$ e assim que $M \in Mod(\mu)$. Para que $M \notin Mod(\psi \circ_c \phi)$ todo modelo $M' \in Mod(\psi \circ_c \phi)$ deve ser tal que $M' <_{Mod(\psi)} M$. Como $Mod((\psi \circ_c \phi) \wedge \mu) \neq \emptyset$, sabemos que existe um $M' \in Mod(\phi \wedge \mu)$, o que contradiz o fato de M ser minimal segundo $\leqslant_{Mod(\psi)}$. Logo, $Mod(\psi \circ_c (\phi \wedge \mu)) \subseteq Mod((\psi \circ_c \phi) \wedge \mu)$, quando $(\psi \circ_c \phi) \wedge \mu$ é satisfazível. E isto completa nossa prova.

□

Capítulo 7

Reparo de especificações parciais via KMTS

Apresentamos neste capítulo a revisão de modelos aplicado a sistemas parcialmente especificados. Uma especificação parcial ocorre, por exemplo, quando um projetista indica que a comunicação entre dois módulos é possível, mas não mandatório. Utilizamos aqui a representação deste tipo de especificação por meio de sistemas de transição denominados *sistemas de transição modal de Kripke.*[1]

7.1 Sistemas de Transições Modais de Kripke como conjuntos de modelos CTL

KMTS são modelos capazes de expressar a indeterminação ou subespecificação de sistemas. São sistemas de transições rotulados com modalidades em suas transições: uma transição pode representar uma transição necessária ou uma transição possível.

Definição 30. Um modelo de transição modal de Kripke (KMTS) é uma tupla $M = \langle AP, S, S_0, R^+, R^-, L\rangle$, onde

- S é um conjunto finito de estados
- $S_0 \subseteq S$ é o estado inicial,
- $R^+ \subseteq S \times S$ e $R^- \subseteq S \times S$ são uma relação de transição tal que $R^+ \subseteq R^-$, e
- $L : S \to 2^{Lit}$ é uma função de rotulação, tal que para todo estado s e $p \in AP$, no máximo um entre p e $\neg p$ pé verdadeiro.

As transições R^+ e R^- são denominadas transições *must* e *may*, respectivamente.

Neste capítulo utilizamos a notação explicita de ponto fixo, assim traduzimos cada fórmula CTL para sua notação em μ-calculus por meio das seguintes equivalências:

$$EF\phi \equiv \mu Z.\phi \vee EXZ$$
$$AF\phi \equiv \mu Z.\phi \vee AXZ$$
$$EG\phi \equiv \upsilon Z.\phi \wedge EXZ$$
$$AG\phi \equiv \upsilon Z.\phi \wedge AXZ$$
$$E[\phi U \phi] \equiv \mu Z.\phi \vee (\phi \wedge EXZ) \text{ e}$$
$$A[\phi U \phi] \equiv \mu Z.\phi \vee (\phi \wedge AXZ).$$

[1] Esse capítulo é baseado no trabalho [**?**], ainda não publicado.

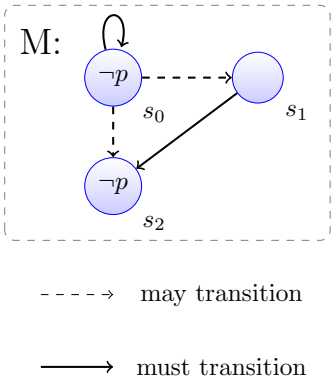


Figura 7.1: *Exemplo de KMTS.*

A semântica abaixo é apresentada em [Gru10] e define a satisfação de fórmulas na lógica μ-calculus sobre modelos KMTS. Em [GLLS07] é apresentado uma definição completa desta semântica.

Definição 31. A semântica de três valores $\|\varphi\|_3^M$ de uma fórmula φ com respeito a um KMTS M é um mapeamento de S para $\{T, F, \bot\}$. Os principais casos deste mapeamento são definidos abaixo

$$[\![\mathtt{l}]\!]_3^M(s) = T \text{ se } \mathtt{l} \in L(s), F \text{ se } \neg\mathtt{l} \in L(s), \bot \text{ caso contrário.}$$

$$[\![AX\varphi]\!]_3^M(s) = \begin{cases} T, & \text{se } \forall t \in S, \text{if } R^-(s,t) \text{ then } [\![\varphi]\!]_3^M(t) = T \\ F, & \text{se } \exists t \in S \text{ tal que } R^+(s,t) \text{ e } [\![\varphi]\!]_3^M(t) = F \\ \bot, & \text{otherwise.} \end{cases}$$

E a dualidade para $EX\varphi$ com a troca de `F` por `T`.

7.1.1 Expansão de KMTS em modelo CTL

Nesta seção definimos formalmente a noção de *expansão* KMTS em conjuntos de estruturas de Kripke, mostrado que esta representação pode ser vista como um modo compacto de representar grandes conjuntos de modelos CTL. Mostramos, contudo, que existem limitações neste tipo de representação.

Definição 32. Seja $M = \langle S, R_{must}, R_{may}, L\rangle$um modelo KMTS, uma expansão KMTS de M, denotada por M_K, é um conjunto de todos as estruturas de Kripke $K' = \langle AP', S', S'_0, R', L'\rangle$ tais que $AP' = AP$, $S' = S$, $S'_0 = S_0$, $R^+ \subseteq R' \subseteq R^-$ e $L(s) \subseteq L'(s)$, para todo $s \in S$.

Uma expansão KMTS pode levar a um conjunto exponencial de estruturas de Kripke, como descrito na Proposição 13. Por outro lado, isso mostra a capacidade deste formalismo de representar compactamente um grande conjunto de modelos CTL em uma única estrutura.

Proposição 13. *Seja $M = \langle S, R_{must}, R_{may}, L\rangle$um conjunto KMTS com $m = |R^- \backslash R^+|$ transições estritamente may e $n = |\{s \in S \mid p \in AP \text{ e } p, \neg p \notin L(s)\}|$ indeterminações em estados. M pode ser expandido para 2^{m+n} diferentes estruturas de Kripke.*

Demonstração. Segue diretamente no número de possíveis combinações de cada indeterminação no KMTS que pode ser estar presente ou não na estrutura de Kripke. □

É importante notar, contudo, que um modelo KMTS pode não ser expressivo o suficiente para representar todos os possíveis conjuntos de modelos CTL, como mostrado na Proposição 14.

Proposição 14. *Seja $K = \{k_1, ..., k_n\}$ um conjunto qualquer de estruturas de Kripke $k_i = \langle AP, S, S_0, R_i, L_i\rangle$. Não necessariamente existe um KMTS $M = \langle S, R_{must}, R_{may}, L\rangle$cuja expansão é igual à K.*

Demonstração. Tome por exemplo $K = \{k_3, k_5\}$ da Figura 7.2. Não existe KMTS $M = \langle\{p\}, \{s_0, s_1, s_2\}, \{s_0\}, \{(s_0, s_0), (s_1, s_2)\}, R_-, L\}\rangle$ que possa ser expandido neste conjunto dado que o formalismo KMTS não provêm um mecanismo condicionalidade entre suas indeterminações. Assim, não podemos expressar em M que as transições (s_0, s_1) e (s_0, s_2) não podem ocorrer ao mesmo tempo. □

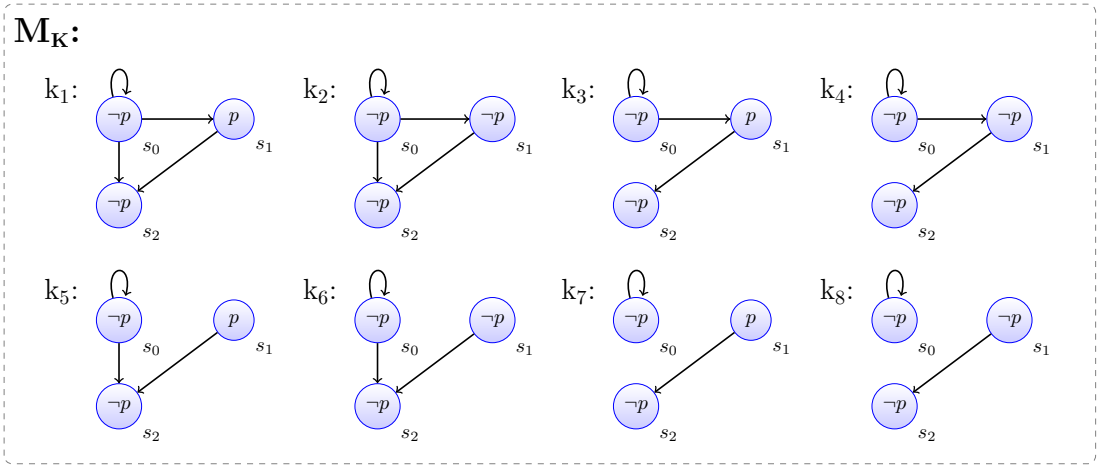


Figura 7.2: *Expansão M_K do modelo da Figura 7.1*

Para representar um conjunto qualquer de estruturas de Kripke temos duas alternativas: (1) associar uma função de seleção a definição de KMTS tal que essa função descreve quais modelos devem ser selecionados dentre os modelos expandidos; (2) considerar que um conjunto de modelos de Kripke é representado por um conjunto de modelo KMTS, sendo nessa alternativa, no pior caso, cada modelo KMTS equivale a um único modelo de Kripke.

Proposição 15. *Seja M um modelo KMTS e $K = \{k_1, ..., k_n\}$ as estruturas de Kripke expandidas de M. Considere s_0 o estado inicial de M, para toda fórmula φ de $\mu-$calculus, se o valor de $\|\varphi\|_3^M(s_0)$ é igual à*

1. *$\bot$, então $\exists k_i, k_j \in K, i \neq j$ tal que $(\|\varphi\|^{k_i}(s_0) = T$ e $\|\varphi\|^{k_j}(s_0) = F$*
2. *T, então $\forall k_i \in K, \|\varphi\|^{k_i}(s_0) = T$*
3. *F, então $\forall k_i \in K, \|\varphi\|^{k_i}(s_0) = F$*

Demonstração. Segue diretamente da definição de semântica de KMTS e da Definição 32. □

7.2 Revisão de Modelos KMTS

Nesta seção iremos definir a operação revisão de modelos KMTS, através da especificação de critério mudança mínima sobre modelos KMTS e mostrando a sua correspondência com as mudanças mínimas sobre conjuntos de modelos de Kripke expandidos de um KMTS. Este critério minimalidade é semelhante ao proposto por [Gue10], mas agora considerando um conjunto diferente de operações primitivas, as quais representam as possibilidades de mudanças nos modelos KMTS, como mostrado abaixo.

P1: Remoção de um par da relação R^-

P2: Remoção de um par da relação R^+

P3: Transformação de um par (s_i, s_j) de R^- para (s_i, s_j) de R^+

P4: Alterar um literal definido em um rótulo estado

P5: Atribuição de um literal a um rótulo de estado, se for previamente indefinido neste

Para as definições abaixo nos consideramos as seguintes notações. X_{Pn} denota o conjunto de mudanças relativas as operações $P_n, 1 \leq n \leq 5$.

Cada mudança em X_{Pn} é representado como (s_i, s_j) ou (s_i, l), onde l é um literal, dependendo se a mudança é relativa a transições ou a rótulos de estados, respectivamente.

Uma mudança X é representado como $X = (X_{P1}, ..., X_{P5})$, onde X_{Pn} pode ser vazio se nenhuma mudança do tipo P_n ocorrer. Dizemos que $X = (X_{P1}, ..., X_{P5}) \subset Y = (Y_{P1}, ..., Y_{P5})$ se para cada $X_{Pn} \subseteq Y_{Pn}$ e pelo menos um $X_{Pi} \subset Y_{Pi}$. A aplicação de X ao modelo A resulta em outro modelo, denotado por $A(X)$. Dizemos que $M, s_0 \models \varphi$ é T, F ou $\perp$ para indicar o resultado de uma verificação de φ em s_0 de M.

Nossa definição de mudança minimal sobre KMTS é baseado nas operações P2 e P4. As operações P1, P3 e P5 serão desconsideradas, visto que entre os modelos de Kripke expandidos de uma KMTS, existem sempre modelos sem as transições removidas por P1, modelos que já possuem as transições transformadas por P3, e modelos que já possuem os valores atribuídos a literais por P5. Neste sentido, essas mudanças são irrelevantes para o resultado da revisão e serão desconsideradas.

Definimos o conceito de mudanças minimais por meio do que chamamos *mudança reduzida* $X/$ de uma mudança X.

Definição 33. Seja $X = (X_{P1}, ..., X_{P5})$, a mudança reduzida $X/$ de X é definida como $X/ = (X_{P2}, X_{P4})$.

Uma mudança reduzida $X/ = (X_{P2}, X_{P4})$ sobre um KMTS M induz mudanças em $K \in M_K$: todo $(s_i, s_j) \in X_{P2}$ induz mudanças (s_i, s_j) do tipo $PU2$ em K, e toda $(s_i, l) \in X_{P4}$ induz mudanças (s_i, l) do tipo $PU3$ em K.

Definição 34. Dada duas mudanças $X1 = (X1_{P1}, ..., X1_{P5})$ e $X2 = (X2_{P1}, ..., X2_{P5})$, $X1 \leq X2$ se e somente se para todo n, $X1/_{Pn} \subseteq X2/_{Pn}$. $X1 < X2$ se e somente se $X1 \leq X2$ e existe pelo menos um n, tal que $X1/_{Pn} \subset X2/_{Pn}$. Se não existe $X2$ tal que $X2 < X1$, $X1$ é dito minimal.

Proposições 16, 17 e 18 mostram que a definição do critério de minimalidade para KMTS corresponde ao critério de minimalidade para conjuntos de modelos de Kripke expandidos a partir deste KMTS, isto é, a revisão de conjuntos de modelos de Kripke pode ser alcançado por meio da revisão do KMTS que representa tal conjunto. A proposição mostra que qualquer mudança em um modelo de Kripke, que pertença à M_K, pode ser alcançada por mudanças em M.

Proposição 16. *Seja M um modelo KMTS, M_K sua expansão correspondente, $K1$ um modelo em M_K e $Y = (Y_{P2}, Y_{P4})$ uma mudança em $K1$. Existe uma mudança X em M tal que $M(X)_K$ contém o modelo $K1(Y)$.*

Demonstração. Construa $X = (X_{P1}, X_{P2}, X_{P3}, X_{P4}, X_{P5})$:

1. X_{P1} contém todas as transições *may* (s_i, s_j) tais que (s_i, s_j) não são transições de $K1$;
2. X_{P2} contém $(s_i, s_j) \in Y_{P2}$ se (s_i, s_j) é uma transição *must* em M, caso contrário (s_i, s_j) é incluída em X_{P1};
3. X_{P3} contém todas as transições *may* (s_i, s_j) que correspondem a transições (s_i, s_j) de $K1$;
4. X_{P4} contém $(s_i, l) \in Y_{P4}$ se l ou $\neg l \in label(s_i)$ em M, caso contrário inclua-o em X_{P5}.

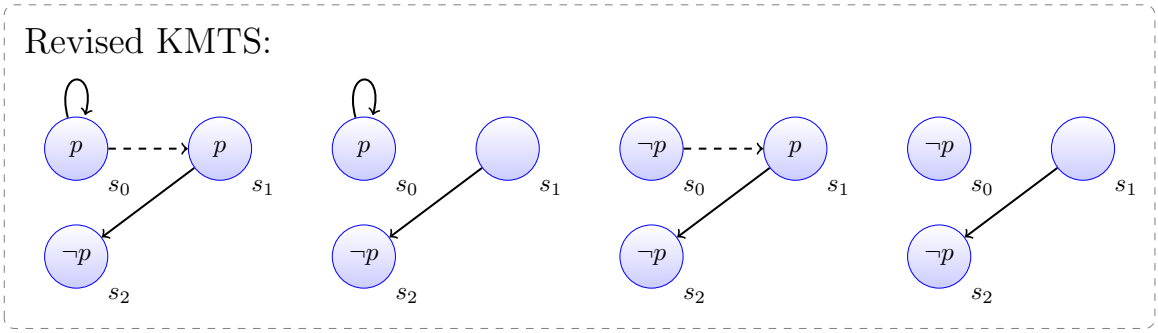
Pegue um modelo $K2$ de M_K que difere de $K1$ de dois modos: 1) para todo $(s_i, l) \in X_{P5}, l \in label(s_i)$ em $K2$; 2) $K2$ não possui transições (s_i, s_j) de Y_{P2} se são transições do tipo *may* em M ($K2$ seguramente existe dado que a expansão de M gera todos os modelos resultantes de todas as possibilidades de transformar indeterminações em M em determinações em modelos de Kripke). Neste caso, $M(X)_K$ contém o modelo $K2(X/)$ que é igual a $K1(Y)$. □

Proposição 17. *Seja M um modelo KMTS e $X = (X_{P1}, X_{P2}, X_{P3}, X_{P4}, X_{P5})$ uma mudança minimal em M. Então, $X/ = (X_{P2}, X_{P4})$ é uma mudança minimal em M_K.*

Demonstração. Suponha que $X/$ não é minimal em M_K, então existe uma mudança $Y = (Y_{P2}, Y_{P4})$ em M_K tal que $Y < X/$. As transições de Y_{P2} não podem ser transições do tipo *may* em M e todos os literais de estados em Y_{P4} são literais definidos em seus respectivos estados de M. Pela Proposição 16 existe uma mudança Z em M construida de Y tal que $Z/ = Y$. Assim, $Z/ < X/$ que implica que $Z < X$, uma contradição. □

Proposição 18. *Seja M um modelo KMTS tal que $M, s_0 \models \varphi$ seja falso, $X = (X_{P1}, X_{P2}, X_{P3}, X_{P4}, X_{P5})$ uma mudança minimal em M tal que $M(X), s_0 \models \varphi$ é verdadeiro, existe um modelo K em M_K tal que $K(X/), s_0 \models \varphi$ é verdadeiro.*

Demonstração. Selecione um modelo K de M_K tal que K não possua transições do tipo X_{P1}, que possua todas as transições de X_{P3} e para todo (s_i, l) em X_{P5}, $l \in L(s_i)$ em K. Para todo K_i de $M(X)_K$, $K_i, s_0 \models \varphi$ é verdadeiro dado que $M(X), s_0 \models \varphi$ é verdadeiro (ver Proposição 15). O modelo $K(X/)$ é um modelo dentre os modelos K_i. □



Minimal changes over M

$$X1 = (\{(s_0, s_2)\}, \emptyset, \emptyset, \{(s_0, p)\}, \{(s_1, p)\})$$
$$X2 = (\{(s_0, s_1), (s_0, s_2)\}, \emptyset, \emptyset, \{(s_0, p)\}, \emptyset)$$
$$X3 = (\{(s_0, s_2)\}, \{(s_0, s_0)\}, \emptyset, \emptyset, \{(s_1, p)\})$$
$$X4 = (\{(s_0, s_1), (s_0, s_2)\}, \{(s_0, s_0)\}, \emptyset, \emptyset, \emptyset)$$

Reduced changes

$$X1/ = (\emptyset, \{(s_0, p)\})$$
$$X2/ = (\emptyset, \{(s_0, p)\})$$
$$X3/ = (\{(s_0, s_0)\}, \emptyset)$$
$$X4/ = (\{(s_0, s_0)\}, \emptyset)$$

Figura 7.3: *Revisão por AXp do KMTS M (Fig. 7.1).*

Figuras 7.3 e 7.4 mostram um exemplo da relação entre mudanças minimais de um KMTS M e mudanças minimais no conjunto de modelo expandidos M_K. Elas apresentam as mudanças minimais possíveis (com as operações primitivas P_1 e P_5 apresentadas no Capítulo 4) em M para satisfazer a propriedade AXp de s_0, bem como suas respectivas mudanças reduzidas que correspondem a mudanças minimais em M_K.

Considere a mudança $X = (\emptyset, \emptyset, \emptyset, \{(s_0, p), (s_2, p)\}, \{(s_1, p)\})$. Essa mudança não é minimal pois, para $X1$ na Figura 7.3, temos que $X1 < X$. Para exemplifica a Proposição 16 considere o modelo K_4 (Figura 7.2) e a mudança $Y = (\emptyset, \{(s_0, p), (s_1, p)\})$, o modelo $K_4(Y) = K_3(X_2/)$. Como exemplo da Proposição 18, considere o modelo K_3 (Figura 7.2) e a mudança X_1 de M (Figura 7.3), $M(X_1), s_0 \models AXp$ é verdadeira e $K_3(X_1/), s_0 \models AXp$ é verdadeira.

No caso onde a verificação do modelo KMTS retorna $\perp$, a revisão KMTS seleciona entre os modelos de Kripke expandidos aqueles consistentes com a propriedade verificada. Este resultado é condizente com o resultado produzido pelo operador $\circ_c$ dado pela Definição 6.3. Quando a verificação de um modelo KMTS retorna falso, as mudanças efetivamente modificam o modelo KMTS. Como dito anteriormente, consideramos apenas um conjunto reduzido de mudanças. Apesar dessas

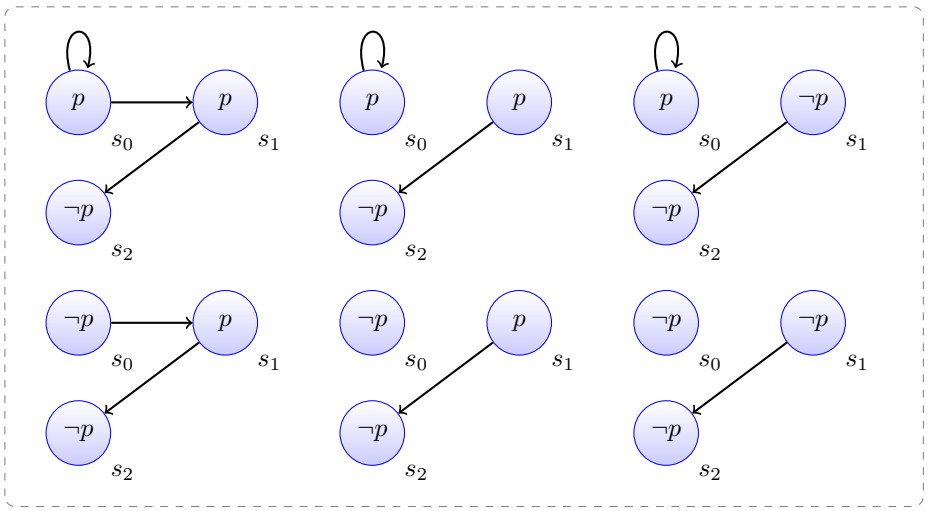


Figura 7.4: *Revisão da expansão de M_K (Figura 7.2) por AXp.*

mudanças serem relativamente restritivas, nossa abordagem é uma evolução no processo de revisão de especificações parciais.

7.3 Implementando revisão de modelos KMTS

A revisão de um modelo KMTS M deve ocorrer quando $M, s_0 \models \varphi$ é $\bot$ ou F. No caso onde a verificação de modelos retorna $\bot$ a revisão consiste de um refinamento de KMTS de modo que o resultado seja um KMTS expansível em apenas modelos de Kripke onde a propriedade desejada é satisfeita. No caso onde a verificação de modelos retorna F, o KMTS deve ser reparado de modo que os possíveis resultados tenham valor T na verificação de modelos.

Nesta seção apresentamos o método de verificação de modelos de três valores proposto por [GLLS07] e nossa proposta de um algoritmo abstrato baseado neste para refinar um modelo KMTS.

7.3.1 Verificação de modelos como um jogo de três valores

O algoritmo de verificação de modelos de três valores para μ-calculus proposto por [GLLS07] consiste de um jogo formal. A verificação de Grumberg faz uso do conceito de estratégia de não-perca para identificar a causa de $\bot$ na verificação de modelos em vez de utilizar a estratégia de vitória. Estes jogos são definidos como um jogo entre dois jogadores $\exists$ e $\forall$, onde o jogador $\exists$ e tenta satisfazer a fórmula e o jogador $\forall$ tenta refutá-la.

O jogo que compõe o algoritmo de verificação consiste em um grafo de configurações do tipo $s \vdash \psi$ onde w é um estado do modelo e ψ é uma subfórmula de φ. Essas configurações são determinadas a partir de uma decomposição da fórmula φ em suas subfórmulas de acordo com as regras apresentadas na Figura 7.5, considerando os estados e transições do modelo KMTS.

Na Figura 7.6 nós mostramos um exemplo de um grafo de configurações de um jogo de verificação de três valores. Uma classificação é do tipo $\exists$ quando ψ é um antecedente de uma regra $\exists$. Configurações $\exists$ são representados por uma elipse no grafo do jogo. Uma configuração é do tipo $\forall$ se ψ é um antecedente de uma regra $\forall$, sendo representada por um retângulo no grafo do jogo. Arestas pontilhadas correspondem a transições genuinamente *may* ($R_- \backslash R_+$). Arestas cheias correspondem a transições *must* ou outras transições geradas pelas regras do jogo que não envolvam transições do modelo verificado.

Os movimentos de um jogador a partir de uma configuração seguem uma estratégia. A estratégia de um jogador é uma função sobre as possíveis configurações do grafo de jogo. Uma estratégia vencedora é uma estratégia que faz o jogador ganhar o jogo independente da estratégia utilizada pelo outro jogador. Quando nenhum dos jogadores ganha o jogo, ambos têm uma estratégia de

não-derrota e o resultado do jogo é $\perp$. Por exemplo, na 7.6, as arestas mais escuras fazem parte de uma estratégia de não-derrota para o jogador $\forall$.

Regras para o jogador $\exists$:

$$\frac{s \vdash \psi_0 \vee \psi_1}{s \vdash \psi_i} : i \in \{0,1\} \qquad \frac{s \vdash EX\psi}{t \vdash \psi} : R^+(s,t) \text{ or } R^-(s,t)$$

$$\frac{s \vdash \eta Z.\psi}{s \vdash Z} : \eta \in \{\mu, \upsilon\} \qquad \frac{s \vdash Z}{s \vdash \psi} : \text{if } f_p(Z) = \eta Z.\psi, \eta \in \{\mu,\upsilon\}, \text{ and } f_p(Z) \text{ é uma única subfórmula identificada por } Z$$

Regras para o jogador $\forall$:

$$\frac{s \vdash \psi_0 \wedge \psi_1}{s \vdash \psi_i} : i \in \{0,1\} \qquad \frac{s \vdash AX\psi}{t \vdash \psi} : R^+(s,t) \text{ or } R^-(s,t)$$

Figura 7.5: *Regras para o jogo de verificação de modelos.*

Uma partida pode ser finita ou infinita, e é definida como uma sequência de configurações $C_0, C_1, ...$ tal que existe uma aresta de C_i para C_{i+1}. Cada configuração de um grafo de jogo é colorida dependendo do resultado de todas as jogadas iniciando desta configuração: colorida com T se o jogador $\exists$ vence; como F se o jogador $\forall$ vence, ou com $\perp$ se nenhum jogador vence. Uma condição necessária para um jogador vencer uma partida é obedecer a restrição que para todos os seus movimentos sejam feitas por meio de transições que não correspondam a transições genuinamente *may* no modelo KMTS avaliado. Outras condições podem determinar o vencedor de uma partida, como apresentado a seguir.

Condição para o jogador $\exists$ vencer uma partida $C_0, C_1, ...$:

1. Existir um $n \in N$ tal que $C_n = t \vdash l$ e o estado t do modelo seja rotulado com l ou
2. Existir um $n \in N$ tal que $C_n = t \vdash AX\psi$ e não existe $t' \in S$ tal que (t, t') é uma transição no modelo, ou
3. a variável mais externa que ocorre com frequência infinitamente é do tipo υ

Condição para o jogador $\forall$ vencer uma partida $C_0, C_1, ...$:

1. Existir um $n \in N$ tal que $C_n = t \vdash l$ e o estado t do modelo seja rotulado com $\neg l$ ou
2. Existir um $n \in N$ tal que $C_n = t \vdash EX\psi$ e não existe $t' \in S$ tal que (t, t') é uma transição no modelo, ou
3. a variável mais externa que ocorre com frequência infinitamente é do tipo μ

Se nenhum dos jogadores vence uma partida, o resultado desta é $\perp$, significando que ambos jogadores tem uma estratégia de não-derrota para a partida. Se um jogador vence um jogo, se ele vence todas as partidas a partir da configuração inicial ($s_0 \vdash \varphi$).

Para calcular o resultado do jogo, colorimos os nós do grafo como T, F ou $\perp$ como explicado na próxima seção, iniciando pelos nós finais (configurações não levam a outras configurações) e então colorindo as configurações que imediatamente as antecedem, repetindo o processo até alcançar a configuração inicial. O resultado do jogo será a cor do nó raiz do grafo de jogo (configuração $s_0 \vdash \varphi$).

A figura 7.6 apresenta um exemplo de grafo de jogo com colorações em suas as configurações (representados pelos símbolos entre parênteses dentro de cada nó do grafo) e como destaque as arestas que pertencem as estratégias de não derrota do jogador $\forall$.

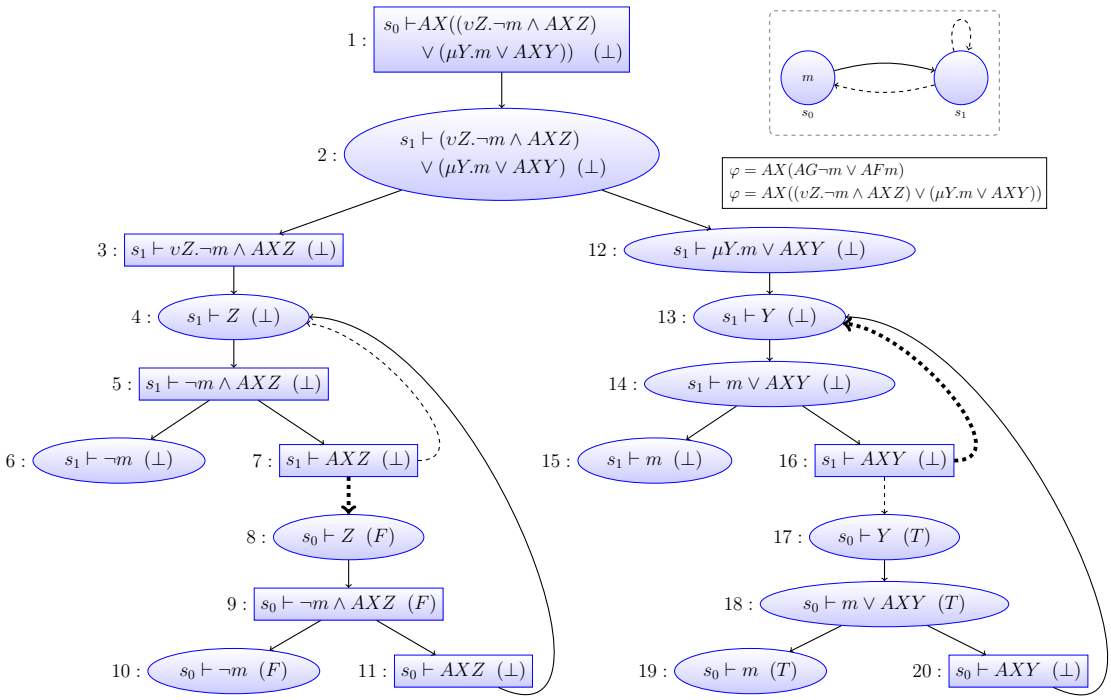


Figura 7.6: *Exemplo de testemunha de falha de uma estratégia de não-derrota para o jogador* $\forall$.

7.3.2 Implementando o Reparo KMTS

Nesta seção desenvolvemos o algoritmo para o refinamento de KMTS para o caso onde $M, s_0 \models \varphi$ é $\bot$, baseado no reparo de um jogo de verificação de três valores. O algoritmo considera estratégias de não-derrota de ambos os jogadores $\forall$ e $\exists$ definidos em [GLLS07] para determinar a testemunha de falha. Nosso algoritmo consiste de uma redução do KMTS para representar apenas estruturas de Kripke que satisfaçam a propriedade φ por eliminação de transições *may* genuínas, transformação destas em transições *must* ou mudando os rótulos indefinidos em estados. Ao final desta seção apresentaremos um esboço de algoritmo para a implementação do reparo quando $M, s_0 \models \varphi$ é falso.

Nesta seção nos referimos as configurações de um jogo como nós. Seja ψ uma subfórmula de φ, definimos *testemunha de falha* no caso onde $M, s_0 \models \psi$ é $\bot$ como uma das seguintes transições que pertençam a estratégia de não-derrota dos jogadores $\forall$ ou $\exists$:

1. uma transição *may* genuína, vindo de um nó do tipo AX colorido com $\bot$, para um nó colorido com F ou $\bot$,
2. uma transição *may* genuína, vindo de um nó do tipo EX colorido com $\bot$, para um nó colorido com T ou $\bot$,
3. uma transição *must*, vindo de um nó do tipo EX colorido com $\bot$, para um nó colorido com $\bot$,
4. uma transição vinda de um nó do tipo $s_i \vdash l \wedge \psi$ para um nó $s_i \vdash l$ colorido com $\bot$,
5. uma transição de um nó $s_i \vdash l \vee \psi$ com uma transição para um nó $s_i \vdash l$ colorido com $\bot$ e uma outra transição para um nó colorido com $\bot$ ou F.

Na Figura 7.6 as transições em desta são exemplos de testemunha de falhas.

De modo a obter todos os modelos de Kripke que satisfaçam a propriedade, todas as testemunhas de falha devem ser consideradas, resultando em diferentes KMTS. Contudo, não é necessário considerar todas as possibilidades de mudanças de modo a gerar todos os modelos de Kripke possíveis, apenas atentar as mudanças nas transições *may* expressas pelo KMTS. Por exemplo, se um

nó $s_0 \vdash EXp$ colorido como $\perp$ tem dois sucessores $s_1 \vdash p$ e $s_2 \vdash p$ no grafo do jogo, ambos coloridos por T, é suficiente mudar apenas uma transição *may* para *must*, porque os KMTS resultantes desta mudança expressam com ambas transições como *must*.

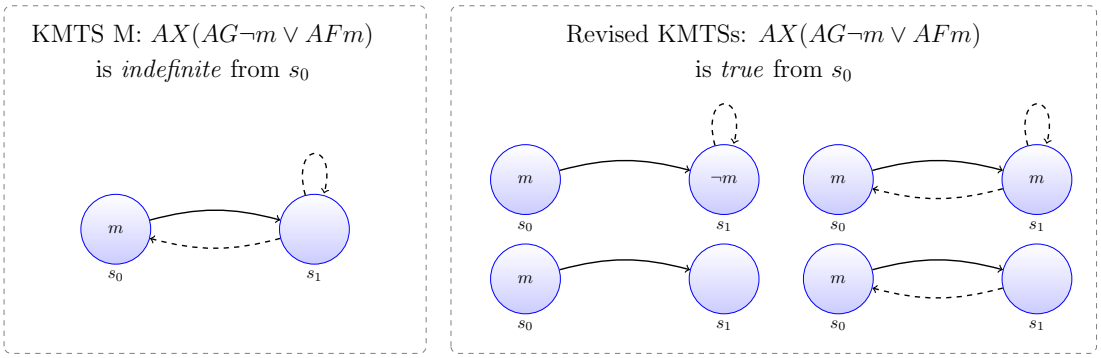


Figura 7.7: *Exemplo de refinamento KMTS.*

O algoritmo *Revision-game* controla o refinamento do modelo M a partir de uma sequência de testemunhas de falha *Fwitness* identificadas pelo procedimento *Check-model*, o qual as determina analisando o grafo do jogo de verificação. O procedimento Refine-game analisa uma testemunha de falha por vez em *Fwitness* até que nenhuma falha exista nesta sequência. Para o jogo da figura 7.6, a sequência *Fwitness* inicializa como $((7,8),(16,13),(14,15))$, onde o par (m,n) representa a transição do nó m para o nó n no grafo do jogo. Outras testemunhas de falha são consideradas pelo procedimento *Refine-game* de modo a complementar a mudança $X = (\{(s_1,s_0)\},\emptyset,\emptyset,\emptyset,\emptyset)$ tais como $(5,6)$ e $(16,13)$. Para este exemplo, o algoritmo de refinamento retorna quatro modelos KMTS (ver Figura 7.7) que satisfazem φ com as seguintes mudanças:

1. $X = (\{(s_1,s_0)\},\emptyset,\emptyset,\emptyset,\{(s_1,\neg m)\})$,
2. $X = (\{(s_1,s_0),(s_1,s_1)\},\emptyset,\emptyset,\emptyset,\emptyset)$,
3. $X = (\{(s_1,s_1)\},\emptyset,\emptyset,\emptyset,\emptyset)$,
4. $X = (\emptyset,\emptyset,\emptyset,\emptyset,\{(s_1,m)\})$.

O algoritmo *Refine-game* controla os possíveis refinamento a partir do conjunto de testemunhas de falhas *Fwitness*. Cada mudança X é usada para modificar e recolorir o grafo do jogo pelo procedimento *Recolor-game*, que por sua vez chama o verificador de modelos de três valores. Uma mudança é feita em um modelo se uma transição (m,n) se: (m,n) corresponde a transição $(s_i,\ s_j)$ em um modelo; todas as transições (r,s) que correspondem à $(s_i,\ s_j)$ devem ser removidas do grafo do jogo; e se os subgrafos do nó s não são mais acessíveis a partir do nó raiz. Essas mudanças não devem mais se consideradas por outras buscas de testemunhas de falha.

Se uma execução de uma verificação de modelos resultar em $\perp$, todas as testemunhas de falha em *Nwitness*, determinadas por *Recolor-game*, são consideradas para complementar X, por meio de chamadas recursivas à *Refine-game*. Quando uma verificação resulta em T o modelo $M(X)$ é retornado, o jogo é restaurado então para o estado anterior (*backtracking*) de modo que outras testemunhas de falha de *Nwitness* e assim todas as possíveis combinações de mudanças sejam consideradas na formação de X.

O algoritmo *Refine-play* gera as mudanças X de acordo com *Failure* $= (m,n)$, a causa da falha relativa ao nó m ou n. Considere o nó m do tipo $(s_i \vdash \psi)$ e n do tipo $(s_j \vdash \chi)$. O nó m pode ser: um nó AX $(s_i \vdash AX\psi)$, um nó EX $(s_i \vdash EX\psi)$, um nó disjuntivo do tipo $s_i \vdash \psi \vee l$, ou um nó conjuntivo do tipo $s_i \vdash \psi \wedge l$. As condições especificadas no algoritmo cobrem os casos descritos abaixo que representam as possibilidades de mudanças requeridas.

Um nó EX é verdadeiro se seu sucessor via transição *must* é colorido com T. Esse nó colorido com F se todos os seus sucessores via transições *may* são coloridos com F. Caso contrário, esse nó

é colorido com $\perp$. Se um nó m é um EX e n é colorido com T e (m, n) é uma transição *may*, ele deve ser transformado em transição *must*. Uma transição para um nó do tipo $s_k \vdash V$ que representa um *loop* no grafo e V é uma variável de maior ponto fixo (isto é, fórmulas do tipo EG) devem também ser alteradas para transições *must*. Um nó AX é verdadeiro se para todos seus sucessores via transições *may* são coloridos com T. Esse nó é falso se um de seus sucessores via transições *must* é colorido com F. Caso contrário esse nó é $\perp$. Se o nó n é colorido como F ou $\perp$, e não é do tipo $s_j \vdash l$, as transições *may* (m, n) devem ser removidas. Se esse nó é do tipo $\perp$ ou $s_j \vdash l$, duas alternativas existem: o rótulo de s_j deve ser mudado para incluir l; ou a transição é removida.

Algorithm 1: Revision-game()

Input: Um modelo KMTS M, uma propriedade φ /* X é declarado como uma variável global */

Output: Conjunto de KMTS resultante de mudanças com $s_0 \vdash \varphi$ colorido como T

```
1  Leia(M, φ) ;
2  Check-model(M, φ, Fwitness); /* retorna um grafo de um jogo de verificação e as possíveis
   testemunhas de falha, caso s0 ⊢ φ tenha a cor ⊥ */
3  if s0 ⊢ φ é colorido como ⊥ then
4  |  repeat
5  |  |  X := () ;
6  |  |  Refine-game(Fwitness, X) ;
7  |  |  Restore-game(head(Fwitness)) ;
8  |  |  Fwitness := tail(Fwitness) ;
9  |  until Fwitness = nil;
10 end
```

Algorithm 2: Refine-game(Fwitness, X)

Input: Fwitness - sequencia de pares (m, n) determinando as testemunhas de falha

Output: Conjunto de KMTS resultantes de mudanças com $s_0 \vdash \varphi$ colorido como T

```
1  Failure := head(Fwitness);
2  Refine-play(Failure, X); /* X contém mudanças a serem feitas */
3  Recolor-game(φ, Nwitness, X); /* outras testemunhas de falha são adicionadas a Nwitness
   caso existam (s0 ⊢ φ é colorida como ⊥) */
4  if s0 ⊢ φ é colorida T then
5  |  Devolva M(X) ;
6  |  Restore-game(Failure) /* o jogo é restaurado removendo a mudança correspondente a
   |  Failure ;
7  else if s0 ⊢ φ é colorido como ⊥ then
8  |  while Nwtiness ≠ nil do
9  |  |  Refine-game(Nwitness, X) ;
10 |  |  Nwitness := tail(Nwitness) ;
11 |  end
12 end
```

Algorithm 3: Refine-play(Change, X)

Input: Failure = (m, n) tal que node m is colored $\perp$ and is of type $s_i \vdash \psi$ and node n is of type $s_j \vdash \chi$
Output: Changes X

if *node* m *is of type* $s_i \vdash EX\psi$ *e* n *is colored* T *or* ($\psi = V$ *e* V *is a variable of type* ν *e* n *is colored* $\perp$ *e* (m, n) *represents a loop in the game graph* **then**
  $X.P_3 := X.P_3 \cup \{(s_i, s_j)\}$;
else if *node* m *is of type* $s_i \vdash AX\psi$ **then**
  if *node* n *is colored* F *or* n *is not of type* $s_j \vdash l$ *and is colored* $\perp$ **then**
    $X.P_1 := X.P_1 \cup \{(s_i, s_j)\}$;
  else if *node* n *is of type* $s_j \vdash l$ *and is colored* $\perp$ **then**
    if $n > 0$ **then** $X.P_5 := X.P_5 \cup \{(s_j, l)\}$;
    else $X.P_1 := X.P_1 \cup \{(s_i, s_j)\}$;
  end
if *node* n *is of type* $s_j \vdash l$ *e* m *is not of type* $s_i \vdash AX\psi$ **then**
  $X.P_5 := X.P_5 \cup \{(s_j, l)\}$;
end

Para implementar o reparo de um KMTS quando a propriedade é inconsistente com o modelo, um algoritmo similar pode ser desenvolvido. Uma estratégia vencedora para o jogador $\forall$ deve ser considerada, em vez de uma estratégia de não-derrota, para a identificação de testemunha de falha. Devemos adicionalmente combiná-la com outras testemunhas de falhas, como nós sem sucessores coloridos com F.

7.4 Notas adicionais sobre o revisão de especificações parciais com KMTS

Como abordado anteriormente, existem conjuntos de estruturas de Kripke não podem ser representados por KMTS únicos (Proposição 14). Uma solução é generalizar revisão KMTS para lidar com conjunto de modelo KMTS, em vez de com um único modelo. Isso pode aumentar a complexidade da revisão, mas com complexidade máxima equivalente a revisão de conjuntos CTL (no pior caso, cada KMTS é expandido para um único modelo CTL). O conjunto de modelos KMTS ainda possui, no caso médio, uma representação mais compacta de especificações parciais, o que permite o desenvolvimento de métodos de revisão mais eficientes.

Para ilustrar essa possibilidade de melhora, tome por exemplo a revisão dos modelos na Figura 7.4, que produziria 32 candidatos ao reparo, que teriam que ser comparado com os 8 modelos iniciais de modo a determinar os modelos minimais, o que envolveria 256 comparações. Essa computação poderia ser ainda mais complexa se envolvesse fórmulas de ponto fixo como EF ou AG, no sentido que o número de candidatos ao reparo é ainda maior. Por outro lado, a revisão KMTS obtém uma solução com 4 KMTS, com quase nenhuma redundância ou modificações desnecessárias.

Capítulo 8

Conclusões

Nesta tese, apresentamos a aplicação da teoria AGM de revisão de crenças sobre os formalismos temporais das lógicas LTL e CTL, com o objetivo de estabelecer uma fundamentação teórica para a aplicação desta teoria no problema do reparo de especificações de sistemas.

Diversos trabalhos abordam o tema. Inicialmente [BEGL99, DZ05, SDE08] propõem o uso de técnicas de IA para, uma vez que uma inconsistência é encontrada em um modelo de sistema, modificar minimamente o modelo. Nesta linha, [HR03, ZD08, SW07, GW10] constroem abordagem teorias de mudanças de crenças, sem, contudo, caracterizar completamente suas operações em termos de postulados ou investigar casos não restritos do problema. Nosso trabalho então busca responder diversas lacunas em aberto sobre o tema.

Apresentamos inicialmente a necessidade de desenvolver duas abordagens distintas para o problema do reparo de especificações inconsistentes. A primeira abordagem é voltada à revisão de conjuntos de fórmulas temporais, semelhante as abordagens clássicas de revisão de crenças. Esta abordagem é adequada ao caso onde a preservação das propriedades esperadas pelo projetista prevalece sobre a preservação da estrutura inicialmente projetada. A segunda abordagem é baseada em modelos, onde devemos modificar sua estrutura para que possa satisfazer uma dada propriedade. Nesta abordagem, a estrutura é o foco de preservação, em contraste com a primeira abordagem.

Mostramos que é impossível realizar a revisão de conjuntos de fórmulas temporais por meio de construções clássicas para revisão e contração de crenças. A razão para isso está na indecidibilidade de se determinar se para um dado conjunto de crenças inconsistente, existe ou não um subconjunto consiste que maximamente preserve as fórmulas originais. Mostramos, porém, que para um subconjunto do problema é possível construir funções de revisão clássicas, desde que assumamos certas propriedades sobre as crenças envolvidas no problema. Apesar de ser uma restrição significativa ao caso geral, o subproblema identificado é ainda expressivo o suficiente para ser combinado com aplicações práticas de verificação formal.

Para a segunda abordagem, utilizamos os mesmos princípios de mudança em modelos em [ZD08] para definir operadores de mudança estrutural em modelos. Em [ZD08], os autores mostram também que seu operador satisfaz um conjunto de postulados de racionalidade. Neste trabalho, contudo, mostramos que nosso operador de revisão de modelos pode ser completamente caracterizado por um conjunto de postulados, i.e., um operador satisfaz nossos postulados se e somente se ele pode ser definido por nossa construção.

Nosso operador de revisão de modelos é de fato equivalente ao operador de atualização proposto por [ZD08] quando lidamos com um único modelo. Mostramos, contudo, que para especificações parciais, em um mundo estático, nosso operador traz resultados mais adequados se comparado a abordagem de [ZD08]. Mostramos também como representar especificações parciais por meio de modelos de Kripke modais, como realizar revisão de modelos sobre esta nova representação e por fim que tais operações também podem ser caracterizadas por postulados de racionalidade.

8.1 Contribuições

Resumimos a seguir as principais contribuições deste trabalho:

- Aprofundamento teórico do problema de reparo de especificações;
- Demonstração da indefinibilidade da construção *partial meet* para conjuntos de fórmulas nas lógicas LTL, CTL e PDL;
- Caracterização de fragmento computável do problema para a aplicação correta de *partial meet*, por meio de restrições no conjunto de fórmulas e no tipo da nova informação;
- Caracterização das operações de revisão de modelos por meio de postulados de racionalidade no estilo AGM;
- Definição de revisão de modelos por meio de sistemas modelais de Kripke (KMTS), como forma de lidar com especificações parciais de sistemas
- Caracterização de revisão de crenças sobre modelos KMTS, também em termos de postulados de racionalidade;

8.2 Trabalhos Futuros

Diversas direções que podem seguir deste trabalho, dentre as quais destacamos:

1. Analisar o problema de revisão de crenças para lógicas temporais mais simples, por exemplo sem ponto-fixo;
2. Analisar o problema de revisão de crenças para o caso geral de lógicas monotônicas e não compactas;
3. Explorar outras construções clássicas para revisão de conjuntos de fórmulas, como contração segura [AM85] e contração *kernel* [Han94];
4. Desenvolvimento de algoritmos para revisão de fórmulas e de modelos, levando em consideração as restrições teóricas apresentadas. Implementá-los como módulos em ferramentas de verificação formal.

Referências Bibliográficas

[AGM85] Carlos E Alchourron, Peter Gärdenfors e David Makinson. On The Logic of Theory Change: Partial Meet Contraction and Revision Functions. *J. Symb. Logic*, 50(2):510–530, 1985. 1, 2, 11, 14, 15, 23

[AM85] Carlos Alchourrón e David Makinson. On The Logic of Theory Change: Safe Contraction. *Studia Logica*, 44:405–422, 1985. 54

[AS85] Bowen Alpern e Fred B. Schneider. Defining liveness. *Information Processing Letters*, 21(4):181–185, oct 1985. 30, 31, 32, 33

[BBF+01] Béatrice Bérard, Michel Bidoit, Alain Finkel, François Laroussinie, Antoine Petit, Laure Petrucci, Philippe Schnoebeln e Pierre McKenzie. *Systems and Software Verification.* 2001. 33

[BEGL99] Francesco Buccafurri, Thomas Eiter, Georg Gottlob e Nicola Leone. Enhancing Model Checking in Verification by AI Techniques. *Artificial Intelligence*, 112(1-2):57–104, aug 1999. 1, 2, 53

[BK08] Christel Baier e Joost-pieter Katoen. *Principles Of Model Checking*, volume 950. 2008. 30

[CBSK12] George Chatzieleftheriou, Borzoo Bonakdarpour, ScottA. Smolka e Panagiotis Katsaros. Abstract Model Repair. Em AlwynE. Goodloe e Suzette Person, editors, *NASA Formal Methods*, volume 7226 of *Lecture Notes in Computer Science*, páginas 341–355. Springer Berlin Heidelberg, 2012. 17

[CCGR99] Alessandro Cimatti, Edmund M Clarke, Fausto Giunchiglia e Marco Roveri. {N}u{S}{M}{V}: A New Symbolic Model Verifier. Em *Proc. of CAV*, páginas 495–499. Springer-Verlag, 1999. 37

[CE82] Edmund M Clarke e E Allen Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. Em Dexter Kozen, editor, *Logics of Programs*, volume 131 of *Lecture Notes in Computer Science*, páginas 52–71. Springer Berlin Heidelberg, 1982. 1, 5, 6, 33

[CES86] Edmund M Clarke, E Allen Emerson e A Prasad Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 8(2):244–263, 1986. 1

[CGP99] Edmund M Clarke, Orna Grumberg e Doron A Peled. *Model checking.* Springer, 1999. 1

[CH12] Max J Cresswell e George Edward Hughes. *A new introduction to modal logic.* Routledge, 2012. 7

[DZ05] Yulin Ding e Yan Zhang. A Logic Approach for LTL System Modification. Em Mohand-Said Hacid, NeilV. Murray, ZbigniewW. Raś e Shusaku Tsumoto, editors, *Foundations*

of Intelligent Systems, volume 3488 of *Lecture Notes in Computer Science*, páginas 435–444. Springer Berlin Heidelberg, 2005. 53

[FL79] Michael J Fischer e Richard E Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences*, 18(2):194–211, 1979. 23, 33

[FW08] Marcelo Finger e Renata Wassermann. Revising Specifications with {C}{T}{L} Properties Using Bounded Model Checking. Em *Brazilian Symposium on Artificial Intelligence*, LNAI. Springer, 2008. 34

[Gär88] Peter Gärdenfors. *Knowledge in flux*. MIT press, 1988. 11, 12, 13

[GHR94] Dov M Gabbay, Ian Hodkinson e Mark Mark A Reynolds. *Temporal logic : mathematical foundations and computational aspects. Volume 1*. Oxford logic guides. Clarendon Press, Oxford, 1994. 6

[GLLS07] Orna Grumberg, Martin Lange, Martin Leucker e Sharon Shoham. When not losing is better than winning: Abstraction and refinement for the full μ-calculus. *Inf. Comput.*, 205(8):1130–1148, 2007. 42, 46, 48

[Gro88] Adam Grove. Two modellings for theory change. *Journal of Philosophical Logic*, 17(2):157–170, 1988. 19

[Gru10] Orna Grumberg. 2-Valued and 3-Valued Abstraction-Refinement in Model Checking. *Logics and Languages for Reliability and Security*, 25(1):105, 2010. 42

[Gue10] Paulo T. Guerra. *Revisão de modelos CTL*. Tese de Doutorado, Universidade de São Paulo, 2010. 2, 43

[GW10] Paulo T. Guerra e Renata Wassermann. Revision of CTL Models. Em Angel Kuri-Morales e Guillermo Simari, editors, *Advances in Artificial Intelligence – IBERAMIA 2010*, volume 6433 of *LNCS*, páginas 153–162. Springer Berlin / Heidelberg, 2010. 2, 17, 18, 53

[Han94] Sven Ove Hansson. Kernel contraction. *The Journal of Symbolic Logic*, 59(03):845–859, 1994. 54

[HKT00] David Harel, Dexter Kozen e Jerzy Tiuryn. *Dynamic logic*. MIT press, 2000. 23, 33

[HR99] Andreas Herzig e Omar Rifi. Propositional belief base update and minimal change. *Artificial Intelligence*, 115(1):107–138, 1999. 2

[HR03] H. Harris e Mark Ryan. Theoretical foundations of updating systems. Em *Proc. IEEE International Conference on Automated Software Engineering*, páginas 291–294, 2003. 53

[HW02] Sven O Hansson e Renata Wassermann. Local change. *Studia Logica*, 70(1):49–76, 2002. 1, 28

[KM91] Hirofumi Katsuno e Alberto O Mendelzon. On the Difference Between Updating a Knowledge Base and Revising it. Em *Proc. of KR*, volume 52, páginas 387–395. Morgan Kaufmann, 1991. 2, 29

[Lam77] L Lamport. Proving the Correctness of Multiprocess Programs. *IEEE Trans. Softw. Eng.*, 3(2):125–143, 1977. 30, 31, 33

[LP85] Orna Lichtenstein e Amir Pnueli. Checking that finite state concurrent programs satisfy their linear specification. *Proceedings of the 12th ACM SIGACT-SIGPLAN symposium on Principles of programming languages - POPL '85*, páginas 97–107, 1985. 1, 5

[Myh60] J Myhill. *Linear Bounded Automata*. Report // University of Pennsylvania. Wright Air Development Division, Air Research and Technology Command, United States Air Force, 1960. 23

[Pnu77] Amir Pnueli. The Temporal Logic of Programs. Em *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, SFCS '77, páginas 46–57, Washington, DC, USA, 1977. IEEE Computer Society. 1, 5

[SDE08] Roopsha Samanta, Jyotirmoy V. Deshmukh e E. Allen Emerson. Automatic Generation of Local Repairs for Boolean Programs. Em *Formal Methods in Computer-Aided Design, 2008. FMCAD '08*, páginas 1–10, nov 2008. 53

[Sip05] M Sipser. *Introduction to the Theory of Computation*. Cengage Learning, 2005. 23, 25

[Sou07] Thiago Carvalho De Sousa. *Revisão de modelos formais de sistemas de estados finitos*. Tese de Doutorado, Universidade de São Paulo, 2007. ix, 37, 38

[SW07] Thiago C Sousa e Renata Wassermann. Handling inconsistencies in {C}{T}{L} model-checking using belief revision. Em *Proc. of the Brazilian Symposium on Formal Methods*, 2007. 2, 17, 37, 53

[VDDV15] Marc Van Zee, Dragan Doder, Mehdi Dastani e Leendert Van Der Torre. A{G}{M} Revision of Beliefs About Action and Time. Em *Proc. International Conference on Artificial Intelligence*, páginas 3250–3256. AAAI Press, 2015. 34

[ZD08] Yan Zhang e Yulin Ding. CTL Model Update for System Modifications. *Journal of Artificial Intelligence Research*, 31(1):113–155, 2008. 2, 3, 17, 19, 35, 53