Universidade de São Paulo

Bacharelado em Ciência da Computação

Trabalho de Conclusão de Curso

Implementação de Algoritmos para Revisão de Modelos CTL

Autor:
Duilio Felix Oliveira da Silva

Supervisora:
Prof. Dra. Renata Wassermann

22 de fevereiro de 2015

Resumo

Revisão de crenças é uma área da inteligência artificial que trata da atitude de um agente inteligente rever ou atualizar sua base de conhecimentos. Neste trabalho, utilizaremos o framework criado por Renato Lundberg[5], o *bcontractor*, para implementar os algoritmos de revisão de modelos CTL descritos na tese de mestrado de Paulo de Tarso[1] e assim, fazer uma ligação entre a teoria e a prática da revisão de crenças.

Sumário

Capítulo 1

Introdução

Primeiramente, faremos uma introdução geral ao trabalho neste capítulo. Deixaremos as explicações mais completas dos assuntos envolvidos para o capítulo 2. No capítulo 3, explicaremos o processo de implementação que utilizamos neste trabalho e mostraremos os resultados de testes decorrentes no capítlo 4.

1.1 Objetivo

Este trabalho é baseado em revisão de crenças e na tese sobre revisão de modelos de lógica de árvores computacionais (CTL) escrita por Paulo de Tarso[1]. Nosso objetivo é implementar os algoritmos sobre revisão de modelos de CTL descritos na tese[1] utilizando um framework próprio para esse tipo de construção, o **bcontractor**, desenvolvido por Renato Lundberg[5].

1.2 Importância do Trabalho

Na área de revisão de crenças existe uma certa segmentação entre a teoria e a prática, ou seja, há pouco material teórico implementado, assim como há poucas implementações com uma base teórica forte. Este problema provavelmente ocorre devido à dificuldade e à especificidade da área e é agravado ao focarmos na teoria voltada aos modelos CTL.

Assim sendo, a implementação de um algoritmo de revisão focado em modelos CTL e com um bom embasamento teórico seria uma boa contribuição para a área. Com a implementação do algoritmo descrito na tese[1], esperamos dar uma transcrição fiel de um algoritmo bem embasado para um algoritmo operante e assim, contribuir com um algoritmo que poderá ser utilizado tanto para estudo, quanto em aplicações reais.

1.3 Utilizações Possíveis

Como dito na seção anterior, além do algoritmo poder ser estudado teoricamente (estudo de eficiência ou do algoritmo em si), ele também poderá ser utilizado em aplicações reais para contribuir na checagem de modelos de projetos (de software ou físicos).

Na seção 2.4 descrevemos um exemplo teórico retirado do capítulo 6 da tese[1] que ilustra uma situação em que o algoritmo implementado neste trabalho poderia depurar um projeto de forma a evitar que situações inesperadas acontecessem e, por consequência, evitando problemas de execução do projeto.

Capítulo 2

Fundamentação Teórica

Antes de falarmos da implementação propriamente dita, um pouco de contexto teórico se faz necessário. Neste capítulo, estão descritos de forma básica e resumida os principais conceitos a serem utilizados neste trabalho.

2.1 Revisão de Crenças

Começaremos falando sobre revisão de crenças, que é a base do trabalho proposto. Revisão de crenças é uma área de inteligência artificial que estuda como um agente inteligente (o programa de interesse) deve se comportar ao receber informações novas para a sua base de conhecimentos.

Em revisão de crenças, o conhecimento de um sistema é representado por uma base de conhecimento, que é um conjunto de asserções lógicas, ou crenças. Um agente consistente pode ter três atitudes em relação a uma crença: Aceitar, rejeitar ou tomá-la como indeterminada.

Normalmente, esse conjunto é mutável, e uma mudança nele pode ser resumida a uma mudança de atitude para determinada crença. Essas mudanças podem ser divididas em três tipos: Quando uma crença α é indeterminada e passa a ser aceita ou rejeitada, a mudança é denominada de **expansão**; quando a crença α é aceita ou rejeitada e passa a ser indeterminada, chama-se **contração**; e quando a crença α muda de aceita para rejeitada ou vice-versa, a mudança é denominada de **revisão**.

2.1.1 Teoria AGM

Para se fazer essas operações, geralmente seguimos os postulados de racionalidade descritos na Teoria AGM, cujo nome vem de seus criadores, Carlos E. Alchourrón, Peter Gändenfors e David Makinson. Estes postulados de-

finem uma série de propriedades que fazem com que as operações sigam o princípio de mudança mínima, ou seja, as operações fazem somente a quantidade necessária de mudanças no conjunto de crenças sem remover ou adicionar informações desnecessariamente.

A Teoria AGM define seis postulados para a contração e outros seis para a revisão de um conjunto de crenças. A seguir, estão enumerados os postulados para um dado conjunto de crenças **K**, uma crença α e um conjunto inconsistente Ψ:

Postulados AGM para Contração

(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*)

Postulados AGM para Revisão

(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) $K * \alpha = \Psi$ se e somente se $\vdash \neg\alpha$ (*consistência*)

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

2.2 CTL

Agora, antes de discutirmos sobre o algoritmo proposto, vamos falar sobre a "linguagem" que ele atende, a Lógica de Árvore Computacional, ou CTL (do inglês, *computation tree logic*).

A CTL é uma lógica modal capaz de representar e raciocinar sobre o tempo, ou seja, ela é capaz de expressar asserções que envolvem condições

temporais. Seu nome vem do fato de que a CTL modela o futuro como ramificações de uma árvore, onde cada nó da árvore é um possível estado futuro.

Sua definição é dada pela seguinte *Backus Naur Form*:

$\phi ::= \top \mid \bot \mid p \mid (\neg\phi) \mid (\phi \vee \phi) \mid (\phi \wedge \phi) \mid (\phi \rightarrow \phi) \mid EX\phi \mid AX\phi \mid EF\phi \mid AF\phi \mid EG\phi \mid AG\phi \mid E[\phi U \phi] \mid A[\phi U \phi]$

onde p é um átomo proposicional, $\neg$, $\vee$, $\wedge$ e $\rightarrow$ são os operadores clássicos de lógica proposicional.

Os operadores temporais da CTL são contituídos de um quantificador de caminhos (E, "existe um caminho", ou A, "para todos os caminhos") seguido de um operador de estado (X, "próximo estado no caminho", U,"até que no caminho", G, "globalmente no caminho", ou F, "futuramente no caminho").

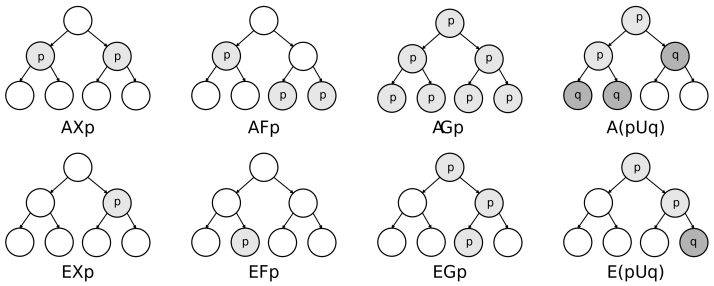

Figura 2.1: Funcionamento dos operadores temporais

2.3 Modelos CTL

Para trabalharmos com alguma fórmula de CTL, podemos criar modelos de estados em que esta fórmula é válida. Esses são os chamados modelos CTL, e que serão representados neste trabalho através das estruturas de Kripke.

As estruturas de Kripke (assim como definido em [1]) são compostas de um conjunto de estados, um conjuto de estados iniciais, uma relação entre seus estados e uma associação entre cada estado e um conjunto de valores proposicionais. No caso de modelos CTL, uma fórmula é satisfeita pelo modelo se ela for verdadeira em cada estado.

A figura a seguir mostra um modelo de três estados com os átomos proposicionais p e q. Este modelo satisfaz, por exemplo, as fórmula $AG(p \vee q)$ e $EF(\neg p \vee \neg q)$.

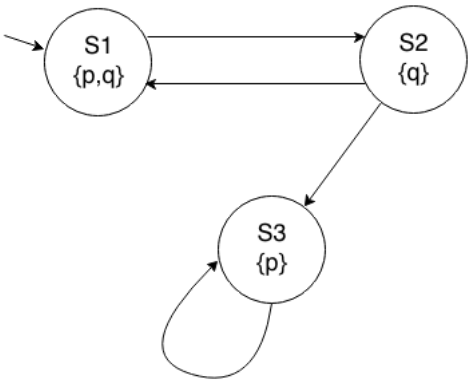

Figura 2.2: Exemplo de uma Estrutura de Kripke

2.4 Revisão de Modelos CTL

Agora, olhando para a parte prática, podemos citar algumas construções dos operadores de revisão e contração como a *partial meet* e o sistema de esferas, mas não falaremos delas neste trabalho. Aqui estamos interessados em uma variante da revisão de crenças, a revisão de modelos. Mais especificamente, a revisão de modelos CTL com os algoritmos propostos na tese[1], que serão visto mais adiante.

O motivo para a criação dessa variante é que em um problema de revisão de crenças é comum usarmos algumas suposições para que a Teoria AGM seja válida, as chamadas suposições AGM. Mas, Hansson e Wasserman[2] provam que apenas compacidade e monotonicidade já são suficientes para que as construções AGM comuns possam ser aplicadas. Contudo, não podemos garantir que as construções AGM típicas possam ser aplicadas na CTL, pois ela não é compacta[1].

Isto motivou a modificação das aborgens típicas e a criação dos algoritmos descritos na tese[1]. Na tese[1], está teoricamente descrito um algoritmo que realiza a revisão de uma base de crenças seguindo os postulados de racionalidade AGM. Este é o algoritmo que foi implementado neste trabalho.

Para ilustrar sua utilização, segue um exemplo retirado do capítulo 6 da tese[1]:

Exemplo 2.1. *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*

[1]Uma lógica é compacta se sempre que $\alpha \in Cn(x)$, existe $x' \subseteq x, x'$ finito tal que $\alpha \in Cn(x')$.

chegar ao consumidor final; e o computador deve eventualmente realizar algumas rotinas de atualização de software.

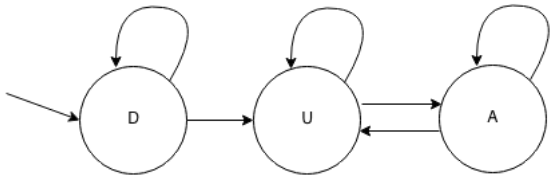

Figura 2.3: Possível modelo para o problema

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:

$\phi \leftrightarrow d \wedge AG(d \rightarrow AX(d \vee u)) \wedge AG(u \rightarrow AX(u \vee a)) \wedge AG(a \rightarrow AX(u \vee a)) \wedge AG((d \wedge \neg u \wedge \neg a) \vee (\neg d \wedge u \wedge \neg a) \vee (\neg d \wedge \neg u \wedge a))$

A figura acima 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 que não satisfazem a seguinte fórmula CTL:

$\varphi \leftrightarrow EF(u) \wedge AF(u)$

O algoritmo implementado neste trabalho poderá ser utilizado para gerar modelos alternativos e consistentes para esse problema através da revisão de modelos CTL. Ou seja, com ele, poderemos reformular o modelo do sistema do exemplo de forma a garantir que seu computador não permaneça infinitamente em atualização, e de forma que o sistema não se desligue (pois já que o computador tem energia infinita, é contra-intuitivo desligá-lo).

2.5 Bcontractor

Para implementar o algoritmo proposto, trabalhamos com o *framework* criado por Renato Lundberg[5], o **bcontractor**. Este framework tem como objetivo dar uma forma prática de programar algoritmos de revisão em Java, assim como uma base para a comparação de algoritmos diferentes.

O bcontractor é uma plataforma genérica que oferece uma ligação entre a parte matemática e a parte computacional da teoria. Isso é feito através de várias estruturas que definem entes como conjutos, sentenças, provadores e outros.

A implementação de operadores de revisão dentro deste ambiente "formaliza", no sentido matemático, o código. Assim, a comparação entre dois operadores distintos que tenham sido implemetados no *bcontractor* se torna muito mais fácil e lógica.

Nele já existem algumas implementações de algoritmos feitas para testes do framework e que também nos serviram como base para o desenvolvimento dos nossos algoritmos. A saber, existem algumas implementações do *partial meet* e uma implementação de revisão para lógica descritiva (ou DL) feita através da OWL (*Web Ontology Language*).

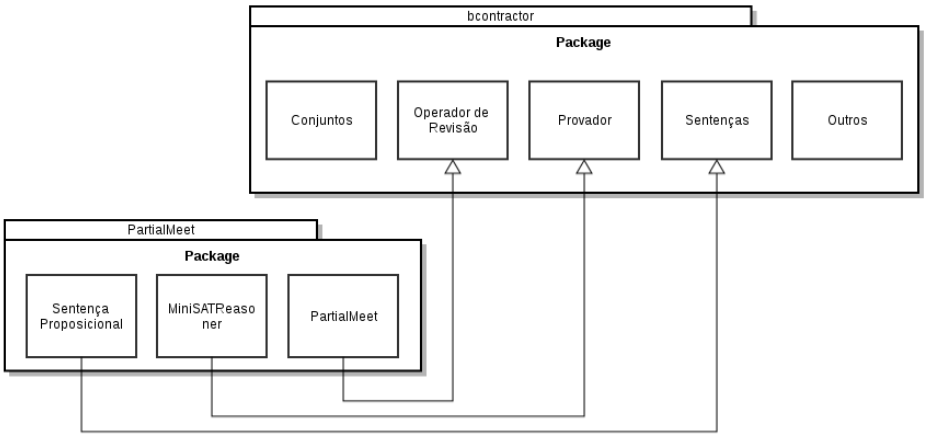

Figura 2.4: Esquematização de uma implementação do Partial Meet no bcontractor

A figura acima esquematiza a implementação do *partial meet* utilizando o **MiniSATReasoner** como provador de fómulas. No nosso trabalho, utilizamos o mesmo estilo de estrutura para realizar a implementação. Os detalhes da implementação estão no capítulo 3.

Capítulo 3

O Trabalho

O intuito do trabalho é implementar os algoritmos de revisão de [1] com o auxílio do *bcontractor*. Para isso, boa parte do ano foi dedicada ao estudo dos tópicos expostos no capítulo 2 e, em especial, ao estudo dos algoritmos em si e da estrutura do *bcontractor* e após isto, foi realizada a implentação dos algoritmos em Java.

Neste capítulo, descrevemos o passo a passo da implementação e detalhamos os algoritmos implementados através de pseudocódigos.

3.1 Visão Geral

Primeiramente, uma análize na estrutura do *bcontractor* nos levou a "dividir" implementação em três etapas:

- Definir classes que reconhececem CTL para formar a classe *Sentence* do *bcontractor*;
- Criar um provador para as sentenças implementadas;
- Implementar o operador de revisão.

Na primeira etapa, implementamos a classe *CTLSentence* a partir da interface *Sentence* dada pelo *bcontractor*. Esta interface nos dá bases linguísticas para trabalhar com sentenças lógicas. No exemplo do *partial meet*, esta classe é implementada para representar sentenças proposicionais através da classe *PropositionalSentence*, enquanto no nosso trabalho, a *CTLSentence* é implementada de forma a reconhecer senteças CTL.

Depois, foi necessário implementar a interface de provadores provida pelo *bcontractor*. Implementar um provador é uma tarefa muito extensa e demorada, então preferimos utilizar um provador já pronto, pois o provador não

é o foco do trabalho. Optamos por utilizar o NuSMV (melhor explicado na próxima seção) como provador, apenas fazendo uma classe que o chame e que receba e trate o seu retorno.

A última etapa foi implementar o algoritmo para o operador de revisão descritos mais adiante sempre respeitando a estrutura de interfaces e classes do *bcontractor*.

Abaixo segue uma esquematização da implementação, no mesmo molde da esquematização do *partial meet* da Figura 2.4:

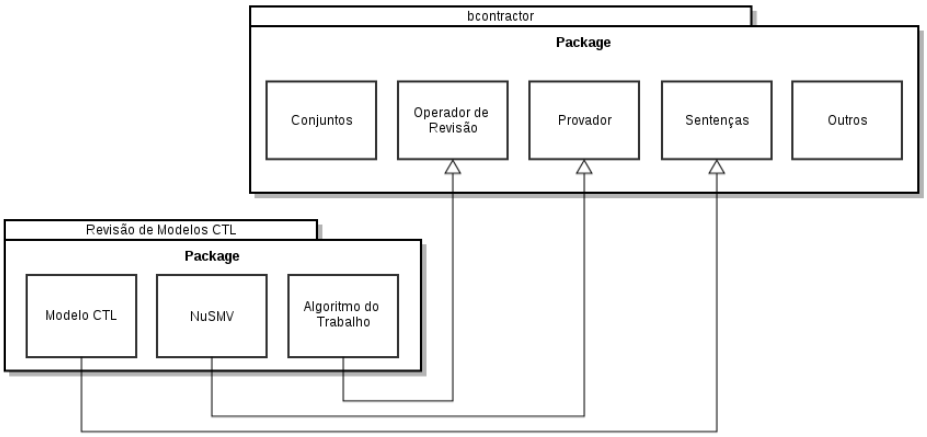

Figura 3.1: Esquematização da implementação de revisão de modelos CTL

3.2 Provador

Para implementar um provador, utilizamos o programa externo NuSMV. O NuSMV é um verificador de modelos que é capaz de apontar se uma dada fórmula CTL é ou não válida em determinado modelo CTL. Porém, ele não realiza a revisão propriamente dita, ele se limita a apontar um caso em que a fórmula não é satisfeita no modelo.

Nesta fase, estudamos a estrutura de entrada do NuSMV e criamos uma classe que faz uma tradução da fórmula em questão para o NuSMV e da resposta do NuSMV para uma resposta booleana reconhecida pelas classes do *bcontractor*.

3.3 Operador de revisão

Assim como as anteriores, o *bcontractor* nos fornece interfaces para os operadores de revisão de crenças (tanto o de contração quanto o de revisão). No trabalho, implementamos apenas o operador de revisão utilizando os algoritmos descritos na tese[1]. Criamos uma classe que implementa a interface *RevisionOperator* fornecida pelo *bcontractor* cujas funções são os algoritmos descritos mais adiante.

Estes algoritmos seguem os postulados de racionalidade AGM e são computacionalmente interessante pois mostram as seguintes propriedades:

- Suas escolhas são determinísticas;
- Suas iterações são feitas sobre conjuntos finitos;
- Utiliza subfunções para o tratamento de cada operador temporal;
- Utiliza recursão de forma a se "aproveitar" do problema.

Todas as escolhas e tomadas de decisão feitas pelo algoritmo são determinísticas, o que permite um bom rastreamento da execução caso seja necessário. Além disso, como o algoritmo só faz iterações sobre conjuntos finitos (como o conjunto de estados ou o conjunto de transições), ele evita um laço infinito.

E como o algoritmo foi subdividido em funções específicas para o tratamento de cada operador temporal, além de se tonar mais legível, o algoritmo também ganha uma certa facilidade em utilizar a recursão algorítmica para tratar a recursão existente na definição da fórmula CTL.

Por último, algumas subfunções utilizam uma ou mais das cinco operações básicas identificadas por Zhang e Ding[3]. Estas são operações pequenas, capazes de derivar as outras operações mais complexas e estão listadas a seguir:

PU1 - Adição de uma relação: Dado $M = (S, R, L)$, um modelo $M' = (S', R', L')$, é obtido de M pela adição de um único novo elemento ao conjunto de relações. Isto é, $S' = S, L' = L$ e $R' = R \cup \{(s_i, s_j)\}$, onde $(s_i, s_j) \notin R$ para dois estados $s_i, s_j \in S$.

PU2 - Remoção de uma relação: Dado $M = (S, R, L)$, um modelo $M' = (S', R', L')$, é obtido de M pela remoção de um único elemento do conjunto de relações. Isto é, $S' = S, L' = L$ e $R' = R - \{(s_i, s_j)\}$, onde $(s_i, s_j) \in R$ para dois estados $s_i, s_j \in S$.

PU3 - Mudança na função de rotulação: Dado $M = (S, R, L)$, um modelo $M' = (S', R', L')$, é obtido de M pela mudança da função de rotulação

para um único estado. Isto é, $S' = S, R' = R, s^* \in S, L'(s^*) \neq L(s^*)$ e $\forall s \in (S - s^*), L'(s) = L(s)$.

PU4 - Adição de um estado: Dado $M = (S, R, L)$, um modelo $M' = (S', R', L')$, é obtido de M pela adição de um único novo elemento ao conjunto de estados. Isto é, $S' = S \cup \{s^*\}$, onde $s^* \notin S, R' = R$ e $\forall s \in S, L'(s) = L(s)$ e $L'(s^*)$ é o conjunto de proposições verdadeiras no estado s^*.

PU5 - Remoção de um estado: Dado $M = (S, R, L)$, um modelo $M' = (S', R', L')$, é obtido de M pela remoção de um único elemento do conjunto de estados. Isto é, $S' = S - \{s^*\}$, onde $s^* \in S$ e $\forall s \in S$, tal que $s \neq s^*, (s, s^*) \notin R$ e $(s^*, s) \notin R, R' = R$ e $\forall s \in S', L'(s) = L(s)$.

Abaixo, seguem os pseudocódigos para a função principal do operador e suas subfunções, assim como descritos na tese[1]:

Algoritmo 1: Revisão$_{CTL}(W, \phi)$

Entrada: W e ϕ, onde $W = \{(M, s_0) | M = (S, R, L)$ e $s_0 \in S\}$ e ϕ é uma fórmula CTL.
Saída: W', representando a revisão de W por ϕ.
se $\exists (M, s_0) \in W$ *tal que* $(M, s_0) \models \phi$ **então**
  $W' \leftarrow \{(M, s_0) | (M, s_0) \in W$ tal que $(M, s_0) \models \phi$
senão
  $W' \leftarrow \emptyset$
  $\phi_t \leftarrow$ tradução de ϕ em termos dos operadores EX, AF e EU
  para cada $(M, s_0) \in W$ **faça**
    caso ϕ_t *seja uma fórmula proposicional*
      $W' \leftarrow W' \cup$ Revisão$_{prop}((M, s_0), \phi_t)$
    caso ϕ_t *seja do tipo* $\neg\phi_1$
      $W' \leftarrow W' \cup$ Revisão$_{\neg}((M, s_0), \neg\phi_1)$
    caso ϕ_t *seja do tipo* $\phi_1 \wedge \phi_2$
      $W' \leftarrow W' \cup$ Revisão$_{\wedge}((M, s_0), \phi_1 \wedge \phi_2)$
    caso ϕ_t *seja do tipo* $\phi_1 \vee \phi_2$
      $W' \leftarrow W' \cup$ Revisão$_{\vee}((M, s_0), \phi_1 \vee \phi_2)$
    caso ϕ_t *seja do tipo* $EX\phi_1$
      $W' \leftarrow W' \cup$ Revisão$_{EX}((M, s_0), EX\phi_1)$
    caso ϕ_t *seja do tipo* $AF\phi_1$
      $W' \leftarrow W' \cup$ Revisão$_{AF}((M, s_0), AF\phi_1)$
    caso ϕ_t *seja do tipo* $E[\phi_1 U \phi_2]$
      $W' \leftarrow W' \cup$ Revisão$_{EU}((M, s_0), E[\phi_1 U \phi_2])$
  fim
  para todo $(M', s'_0) \in W'$ *tal que* $\exists (M'', s''_0) \in W'$ *e* $M'' <_W M'$ **faça**
    $W' \leftarrow W' - \{(M', s'_0)\}$
  fim
fim
retorna W'

Este primeiro algoritmo é a função principal. Seu trabalho é, basicamente, decidir qual é a subfunção mais adequada a se chamar. Ele tem como entrada um conjunto de modelos CTL e uma fórmula CTL. Sua saída é um conjunto de modelos que satisfaz a fórmula ϕ, minimamente próximo do conjunto original.

Algoritmo 2: Revisão$_{EX}((M, s_0), EX\phi)$

Entrada: (M, s_0) e $EX\phi$, onde $M = (S, R, L)$, $s_0 \in S$ e $(M, s_0) \not\models EX\phi$.
Saída: W tal que $\forall(M', s'_0) \in W$, $(M', s'_0) \models EX\phi$.
$W \leftarrow \emptyset$
para todo $s \in S$ *tal que* $(s_0, s) \in R$ **faça**
  $W \leftarrow W \cup$ Revisão$_{CTL}(\{(M, s)\}, \phi)$
fim
para todo $s \in S$ *tal que* $(M, s) \models \phi$ **faça**
  Aplique PU1 para criar um novo modelo $M' = (S', R', L')$:
    $S' = S, R' = R \cup \{(s_0, s)\}, \forall s \in S, L'(s) = L(s)$.
  $W \leftarrow W \cup \{M'\}$
fim
Aplique PU4 e PU1 para criar um novo modelo $M' = (S', R', L')$:
  $S' = S \cup \{(s^*)\}, R' = R \cup \{(S_0, s^*)\}, \forall s \in S, L'(s) = L(s), L'(s^*) = \{\#\}$.
$W \leftarrow W \cup$ Revisão$_{CTL}(\{(M, s^*)\}, \phi)$
$W \leftarrow$ Expanda(W) **retorna** W

O algoritmo de revisão para EX faz três tipos de alteração: O primeiro laço modifica os vizinhos de s_0 para que estes satisfaçam ϕ. O segundo laço adiciona transições de s_0 para estados de M que já satisfazem ϕ. Por último, um novo estado onde ϕ é satisfazível é criado e é feita uma transição de s_0 para este estado.

Algoritmo 3: Revisão$_{AF}((M, s_0), AF\phi)$

Entrada: (M, s_0) e $AF\phi$, onde $M = (S, R, L), s_0 \in S$ e $(M, s_0) \not\models AF\phi$.
Saída: W tal que $\forall(M', s'_0) \in W, (M', s'_0 \models AF\phi$.
$W \leftarrow \emptyset$
$S_\pi \leftarrow \{S | S =$ estados de $\pi = [s_0, s_1, ...]$ em M, tal que $\forall s_i \in \pi, (M, s_i) \not\models \phi\}$
$S_\star \leftarrow \{Q_\star | \forall S \in S_\phi, Q_\star \cap S \neq \emptyset,$ e $\forall S' \subset Q_\star, \exists S \in S_\pi$ tal que $S \cap S' = \emptyset\}$
para todo $Q_\star \in S_\star$ **faça**
  $W' \leftarrow \{(M, s_0)\}$
  enquanto $\exists(M', s'_0 \in W'$ *e* $\exists q_\star \in Q_\star$ *tal que* $(M', q_\star) \not\models \phi$ **faça**
    $W' \leftarrow W' - \{M', s'_0)\}$
    $W' \leftarrow W' \cup$ Revisão$_{CTL}(\{(M', q_\star)\}, \phi)$
  fim
  $W \leftarrow W \cup W'$
fim
Aplique PU2 para criar $M' = (S', R', L')$:
  $S' = S, R' = R - \{(s_0, s) | (s_0, s) \in R$ e $(M, s) \not\models AF\phi\}, \forall s \in S, L'(s) = L(s)$.
$W \leftarrow W \cup \{(M', s'_0)\}$.
retorna W

Nesta função, cada elemento de $S_\star$ é um conjunto mínimo de estados que devem ser modificados para que $AF\phi$ seja satisfeito. O algoritmo realiza

essas modificações e cria um novo modelo que satisfaz $AF\phi$ retirando as transições do estado inicial que levam a caminhos inconsistentes.

Algoritmo 4: Revisão$_{EU}((M, s_0), E[\phi_1 U \phi_2])$

Entrada: (M, s_0) e $E[\phi_1 U \phi_2]$, onde $M = (S, R, L)$, $s_0 \in S$ e $(M, s_0) \not\models E[\phi_1 U \phi_2]$.
Saída: W tal que $\forall (M', s_0') \in W, (M', s_0') \models E[\phi_1 U \phi_2]$.
$W \leftarrow \emptyset$
$S_\star \leftarrow \{s | s \in S$ tal que $\exists \pi = [s_0, ..., s, ...]$ em M, onde $\forall s_i < s, (M, s_i) \models \phi_1\}$
para todo $s_\star \in S_\star$ **faça**
  $W \leftarrow W \cup$ Revisão$_{CTL}(\{(M, s_\star)\}, \phi_2)$
  se $(M, s_\star) \not\models \phi_1$ *e* $\exists (s_\star, s) \in R$ *tal que* $(M, s) \models E[\phi_1 U \phi_2]$ **então**
    $W \leftarrow W \cup$ Revisão$_{CTL}(\{(M, s_\star)\}, \phi_1)$
  fim
  se $(M, s_\star) \models \phi_1$ **então**
    para todo $s \in S$ *tal que* $(M, s) \models E[\phi_1 U \phi_2]$ **faça**
      Aplique PU1 para criar um novo modelo $M' = (S', R', L')$, tal que
        $S' = S, R' = R \cup \{(s_\star, s)\}, \forall s \in S, L'(s) = L(s)$.
      $W \leftarrow W \cup \{M'\}$
    fim
    Aplique PU4 e PU1 para criar um novo modelo $M' = (S', R', L')$:
      $S' = S \cup \{(s^*)\}, R' = R \cup \{(s_\star, s^*)\} \forall s \in S, L'(s) = L(s), L'(s^*) = \{\#\}$.
    $W \leftarrow W \cup$ Revisão$_{CTL}(\{(M, s^*)\}, \phi_2)$
  fim
fim
$W \leftarrow$ Expanda(W)
retorna W

Neste algoritmo, são utilizadas quatro modificações diferentes sobre a entrada. $S_\star$ é o conjunto de estados s para os quais existe um caminho entre s_0 e s onde todos os antecessores de s satisfazem ϕ_1. Para cada $s \in S_\star$ modificamos s para satisfazer ϕ_2 para gerar um caminho consistente com $E[\phi_1 U \phi_2]$; se ϕ_1 não vale em s, mas este é o único empecilho à existência de um caminho consistente, modificamos s para que satisfaça ϕ_1; se ϕ_1 vale em s, adicionamos transições de s a estados onde $E[\phi_1 U \phi_2]$ já vale; e se ϕ_1 vale em s, criamos uma transição de s a um novo estado, fazendo com que ϕ_2 seja satisfazível neste estado.

Algoritmo 5: Revisão$_{prop}((M, s_0), \phi)$

Entrada: (M, s_0) e ϕ, onde $M = (S, R, L)$, $s_0 \in S$
Saída: W tal que $\forall (M', s'_0) \in W$, $L'(s'_0) \models \emptyset$
$W \leftarrow \emptyset$
para cada $P \subseteq 2^{AP}$ *tal que* $P \models \emptyset$ *e* $diff(L(s_0), P)$ *é minimal* **faça**
  Aplique PU3 para criar um novo modelo $M' = (S', R', L')$:
    $S' = S, R' = R, \forall s \in S - \{s_0\}, L'(s) = L(s)$ e $L'(s_0) = P$.
  $W \leftarrow W \cup \{M'\}$
fim
retorna W

Esta é a função que finaliza a recursão. Ela busca mudanças mínimas nas rotulações dos estados para fazer com que ϕ seja satisfeita.

Algoritmo 6: Revisão$_{\wedge}((M, s_0), \phi_1 \wedge \phi_2)$

Entrada: (M, s_0), ϕ_1 e ϕ_2, onde $M = (S, R, L), s_0 \in S$ e $(M, s_0) \not\models \phi_1 \wedge \phi_2$.
Saída: W tal que $\forall (M', s'_0) \in W, (M', s'_0) \models \phi_1 \wedge \phi_1$.
$W \leftarrow$ Revisão$_{CTL}(\{(M, s_0)\}, \phi_1)$
$W \leftarrow$ Revisão$_{CTL}(W, \phi_2)$, restrito por ϕ_1
retorna W

A revisão por conjunção revisa o conjunto de modelos por ϕ_1 e, depois, revisa seu resultado por ϕ_2 garantindo que ϕ_1 continue satisfazível.

Algoritmo 7: Revisão$_{\vee}((M, s_0), \phi_1 \vee \phi_2)$

Entrada: (M, s_0), ϕ_1 e ϕ_2, onde $M = (S, R, L), s_0 \in S$ e $(M, s_0) \not\models \phi_1 \vee \phi_2$.
Saída: W tal que $\forall (M', s'_0) \in W, (M', s'_0) \models \phi_1 \vee \phi_1$.
$W \leftarrow \emptyset$
$W \leftarrow W \cup$ Revisão$_{CTL}(\{(M, s_0)\}, \phi_1)$
$W \leftarrow W \cup$ Revisão$_{CTL}(\{(M, s_0)\}, \phi_2)$
retorna W

O resultado desta função é uma união simples de revisões independentes do conjunto de entrada por ϕ_1 e ϕ_2 separadamente.

Para tratar negações, foram criadas funções distintas, pois queremos modelos que não satisfazem ϕ e que sejam obtidos por modificações mínimas no modelo de entrada.

Algoritmo 8: Revisão$_{\neg}((M, s_0), \phi_1 \neg \phi_2)$

Entrada: (M, s_0) e ϕ, onde $M = (S, R, L), s_0 \in S$ e $(M, s_0) \models \phi$.
Saída: W tal que $\forall (M', s'_0) \in W, (M', s'_0) \not\models \phi$.
caso *ϕ seja uma fórmula proposicional:*
  retorna *Revisão$_{prop}$*$((M, s_0), \neg\phi)$

caso *ϕ seja do tipo $\phi_1 \vee \phi_2$:*
  retorna *Revisão$_{\wedge}$*$((M, s_0), \neg\phi_1 \wedge \neg\phi_2)$

caso *ϕ seja do tipo $\phi_1 \wedge \phi_2$:*
  retorna *Revisão$_{\vee}$*$((M, s_0), \neg\phi_1 \vee \neg\phi_2)$

caso *ϕ seja do tipo $EX\phi_1$:*
  retorna *Revisão$_{\neg EX}$*$((M, s_0), EX\phi_1)$

caso *ϕ seja do tipo $AF\phi_1$:*
  retorna *Revisão$_{\neg AF}$*$((M, s_0), AF\phi_1)$

caso *ϕ seja do tipo $E[\phi_1 U \phi_2]$:*
  retorna *Revisão$_{\neg EU}$*$((M, s_0), E[\phi_1 U \phi_2])$

Esta função tem como papel, chamar as subfunções de negação corretas dependendo da estrutura da fórmula ϕ. As subfunções de negação operam de maneira "inversa" às suas subfunções afirmativas correspondentes.

Algoritmo 9: Revisão$_{\neg EX}((M, s_0), EX\phi)$

Entrada: (M, s_0) e $EX\phi$, onde $M = (S, R, L), s_0 \in S$ e $(M, s_0) \models EX\phi$.
Saída: W tal que $\forall (M', s'_0) \in W, (M', s'_0) \not\models EX\phi$.
$W \leftarrow (M, s_0)$
enquanto $\exists (M', s'_0) \in W$ *tal que* $(M', s'_0) \models EX\phi$ **faça**
  $W \leftarrow W - \{(M', s'_0)\}$
  Selecione $s_\star$ tal que $(s'_0, s_\star) \in R'$ e $(M', s_\star) \models \phi$
  $W \leftarrow W \cup$ Revisão$_{CTL}(\{(M', s_\star)\}, \neg\phi)$
  Aplique PU2 para criar $M'' = (S'', R'', L'')$:
    $S'' = S', R'' = R - \{(s'_0, s_\star)\}, \forall s' \in S', L''(s') = L'(s')$.
  $W \leftarrow W \cup \{(M'', s''_0)\}$
fim
retorna W

O objetivo desta função é eliminar a existência de estados vizinhos à s_0 que satisfaçam ϕ.

Algoritmo 10: Revisão$_{\neg AF}((M, s_0), AF\phi)$

Entrada: (M, s_0) e $AF\phi$, onde $M = (S, R, L), s_0 \in S$ e $(M, s_0) \models AF\phi$.
Saída: W tal que $\forall (M', s_0') \in W, (M', s_0') \not\models AF\phi$.

$W \leftarrow (M, s_0)$
se $(M, s_0) \models \phi$ **então**
  $W' \leftarrow$ Revisão$_{CTL}(\{(M, s_0)\}, \neg\phi)$
  $W \leftarrow$ Revisão$_{CTL}(W', \neg AF\phi)$
senão
  $S_\pi = \{Q_\pi | \exists \pi = [q_0, q_1, ...]$ em M tal que $Q_\pi \subseteq \pi, \forall q \in Q_\pi, (M, q) \models \phi$
    e $\forall q' \in \pi - Q_\pi, (M, q') \not\models \phi\}$
  $S_{min} = \{Q_\pi | Q_\pi \in S_\pi$ e é $\subseteq$-mínimo em $S_\pi\}$
  para todo $Q_{min} \in S_{min}$ **faça**
    $W' \leftarrow \{(M, s_0)\}$
    enquanto $\exists (M', s_0') \in W'$ *e* $\exists q' \in Q_{min}$, *tal que* $(M', q') \models \phi$ **faça**
      $W' \leftarrow W' - \{(M', s_0')\}$
      $W' \leftarrow W' \cup$ Revisão$_{CTL}(\{(M', s')\}, \neg\phi)$
    fim
    $W \leftarrow W \cup W'$
  fim
  $S_\star = \{s | s \in S$ tal que $\exists \pi = [s_0, s_1, ..., s, ...]$ em $M, (M, s) \not\models \phi$
    e $\forall s_i < s, (M, s_1) \not\models \phi\}$
  para todo $s_\star \in S_\star$ **faça**
    para todo $s \in S$ *tal que* $(M, s) \not\models AF\phi$ **faça**
      Aplique PU1 para criar $M' = (S', R', L')$:
        $S' = S, R' = R \cup \{(s_\star, s)\}, \forall s \in S, L'(s) = L(s)$.
      $W \leftarrow W \cup \{M'\}$
    fim
    Aplique PU4 e PU1 para criar um novo modelo $M' = (S', R', L')$:
      $S' = S \cup \{(s^*)\}, R' = R \cup \{(s_\star, s^*)\}, \forall s \in S, L'(s) = L(s), L'(s^*) = \{\#\}$.
    $W \leftarrow W \cup$ Revisão$_{CTL}(\{(M', s^*)\}, \neg\phi)$
  fim
fim
retorna W

Aqui, garantimos que $AF\phi$ não é satisfeita em pelo menos um dos caminhos iniciados em s_0

Algoritmo 11: Revisão$_{\neg EU}((M, s_0), E[\phi_1 U \phi_2])$

Entrada: (M, s_0) e $E[\phi_1 U \phi_2]$, onde $M = (S, R, L)$, $s_0 \in S$ e $(M, s_0) \models E[\phi_1 U \phi_2]$.
Saída: W tal que $\forall (M', s_)') \in W, (M', s_)') \not\models E[\phi_1 U \phi_2]$.

$W \leftarrow (M, s_0)$
se $(M, s_0) \models \phi_2$ **então**
  $W' \leftarrow$ Revisão$_{CTL}(\{(M, s_0)\}, \neg\phi_2)$
  $W \leftarrow$ Revisão$_{CTL}(W', E[\phi_1 U \phi_2])$
senão
  $S_\pi = \{Q_\pi | Q_\pi =$ estados q_i de $\pi = [q_0, q_1, ..., q_j, ...]$ em M tal que
    $(M, q_j) \models \phi_2$ e $\forall q_i < q_j, (M, q_i) \models \phi_1 \wedge \neg\phi_2\}$
  $S_\star = \{Q_\star | \forall Q_\pi \in S_\pi, Q_\star \cap S_\pi \neq \emptyset$ e $\forall Q'_\star \subset Q_\star, \exists Q'_\pi \in S_\pi$ tal que $S'_\star \cap S'_\pi = \emptyset\}$
  para todo $Q_\star \in \S_\star$ **faça**
    $W' \leftarrow \{(M, s_0)\}$
    enquanto $\exists (M', s'_0) \in W'$ *e* $\exists q_\star \in Q_\star$, *tal que* $(M', q_\star) \models \phi$ **faça**
      $W' \leftarrow W' - \{(M', s'_0)\}$
      $W' \leftarrow W' \cup$ Revisão$_{CTL}(\{(M', s_\star)\}, \neg\phi_1)$
    fim
    $W \leftarrow W \cup W'$
  fim
  Aplique PU2 para criar $M' = (S', R', L') : S' = S$,
    $R' = R - \{(s_0, s_*) | (s_0, s_*) \in R$ e $(M, s_*) \models E[\phi_1 U \phi_2]\}, \forall s \in S, L'(s) = L(s)$.
  $W \leftarrow W \cup \{(M', s'_0)\}$
fim
retorna W

Esta última função falsifica a fórmula $E[\phi_1 U \phi_2]$ no estado s_0 do modelo M. Ela altera um conjunto minimal de estados para que essa fórmula não seja satisfeita e retira a alcançabilidade dos segmentos que a satisfazem.

Todas as funções acima são feitas de forma a proteger a minimalidade das alterações dos modelos originais.

Capítulo 4

Considerações Finais

4.1 Conclusão

Este trabalho foi de extenso aprendizado e mostrou o quanto há para ser feito na área de revisão de crenças, porém o *bcontractor* se provou de grande utilidade para implementações de algoritmos desta área.

Facilitar a construção de algoritmos enquanto garante uma boa ligação destes com a teoria é uma grande contribuição para a área e a implementação proposta por este trabalho comprova a eficácia prática do framework.

Por fim, a implementação proposta também abre a possibilidade de estudos futuros sobre o algoritmo e consequentemente, a possibilidade de se fazer melhorias nele.

4.2 Trabalhos Futuros

Ainda há muito a ser feito com este projeto. Primeiramente, a implementação do algoritmo ainda precisa ser completada para entrar em funcionamento.

Além disso, também podem ser feitos testes de desempenho e eficiência para o algoritmo e para a implementação e para melhorar o desempenho, um provador nativo poderia ser implementado.

Por fim, o ideal seria fazer uma integração do algoritmo com algum software para realizar as revisões de modelos (ou fazer a implementação desse software, se necessário).

Referências Bibliográficas

[1] Paulo de Tarso Guerra Oliveira. Revisão de Modelos CTL. *São Paulo: Instituto de Matemática e Estatística, Universidade de São Paulo*, 2010. Dissertação de Mestrado em Ciência da Computação.

[2] Sven O. Handdon e Renata Wassermann. Local change. *Studia Logica*, 2002.

[3] Yan Zhang e Yulin Ding. Ctl model update for system modifications. *Journal of Artificial Intelligence Research, 31(1):113-155*, 2008.

[4] Peter Gärdenfors. *Knowledge in Flux: Modeling the Dynamics of Epistemic States*. College Publications, 2008.

[5] Renato Urquiza Lundberg. Análise Empírica de Algoritmos de Revisão de Crenças. *São Paulo: Instituto de Matemática e Estatística, Universidade de São Paulo*, 2013. Dissertação de Mestrado em Ciência da Computação.