MAC0499 - Trabalho de Formatura Supervisionado

Revisão de crenças no fragmento universal da CTL usando verificação de modelos limitada

Por:
Bruno Vercelino da Hora
Orientador:
Marcelo Finger

Novembro de 2009

Sumário

Resumo

Uma importante etapa do desenvolvimento de software é o de levantamento e análise dos requisitos. Porém, durante esta etapa podem ocorrer inconsistências que prejudicarão o andamento do projeto. Além disso, após finalizada a especificação, o cliente pode querer acrescentar ou modificar as funcionalidades do sistema. Tudo isso requer que a especificação do software seja revista, mas isso é altamente custoso, tornando um processo automatizado necessário para simplificar tal revisão.

O objetivo deste trabalho é utilizar o processo de "Revisão de Crenças" e "Verificação de Modelos" para avaliar especificações de um projeto utilizando CTL (Computation Tree Logic), uma linguagem formal, procurando inconsistências e revisá-las sugerindo sugestões de mudanças na especificação. Iremos seguir a linha proposta por [4], utilizando a implementação de [6].

Iniciamos descrevendo a base teórica por trás do problema. Na segunda seção descreveremos a teoria de revisão de crenças, na terceira descreveremos a lógica temporal que usaremos para representar as propriedades, e na quarta descreveremos a técnica de verificação de modelos e a abordagem escolhida. Na quinta seção iremos descrever de forma detalhada o algoritmo proposto para solucionar o problema, e na sexta seção iremos apresentar um exemplo para esclarecer o algoritmo. Por fim na sétima seção apresentamos algumas conclusões e resultados obtidos através do trabalho.

1 Introdução

Uma das etapas do desenvolvimento de software é o de levantamento e análise de requisitos. Nela são observadas as funcionalidades necessárias e desejadas do software, tendo sua importância pois qualquer erro, inconsistência ou esquecimento pode acarretar em mudanças em todo o projeto, atrasando a entrega e custando ainda mais. No entanto, na prática é difícil conseguir evitar tais reajustes, devido a dificuldade de avaliar o conjunto de requisitos, principalmente conforme a complexidade aumenta. Outro detalhe é que durante o processo de desenvolvimento, o usuário pode querer acrescentar ou modificar algum requisito, sendo necessário uma reavaliação da especificação. Para isso foram desenvolvidas técnicas para analisar uma especificação e verificar por erros e até sugerir mudanças, sendo que um processo automático iria diminuir os custos e o tempo desprendido nesta operação.

Essa técnica é conhecida como Revisão de Crenças, que tem como objetivo manter a consistência de um conjunto de crenças (conjunto de informações) ao ser adicionado alguma informação nova. Assim, a partir do modelo de uma especificação, normalmente em uma linguagem formal, verificamos uma propriedade, através do processo de verificação de modelos , resultando em sugestões para que o sistema continue consistente.

1.1 Objetivos do trabalho

O objetivo deste trabalho é utilizar o processo de "Revisão de Crenças"e "Verificação de Modelos"para avaliar especificações de um projeto utilizando CTL (Computation Tree Logic), uma linguagem formal, procurando inconsistências e revisá-las sugerindo sugestões na especificação.

2 Revisão de crenças

A revisão de crenças teve como fundamento a necessidade de modelar o comportamento de bases de conhecimento que ao receberem novas informações se tornam inconsistentes. Assim, o objetivo principal da revisão de crenças é manter o conjunto de crenças sempre consistente. As bases da teoria de Revisão de Crenças foram desenvolvidas por Alchourrón, Gärdenfors e Makinson [1], no início da década de 80, onde propuseram postulados que descrevem as propriedades formais que um processo de revisão deve obedecer. O trabalho é normalmente conhecido como *paradigmas AGM*, devido as iniciais dos autores. Um dos principais fundamentos da teoria de revisão de crenças, é o *Princípio da Mudança Mínima*, ou *PMM*, que determina que a mudança no conjunto original de crenças seja a mínima possível, ou seja, reter o máximo de informação possível.

Existem três tipos de operações, e cada uma delas possuem seus postulados afim de que o *Princípio da Mudança Mínima* seja satisfeito quando uma mudança é realizada.

1. **Expansão ($K + \alpha$):** Uma informação (crença) consistente α, juntamente com suas conseqüências lógicas é adicionada ao conjunto.
2. **Contração ($K - \alpha$):** A informação (crença) α é abandonada. Como o conjunto K é logicamente fechado, é possível que seja necessário abandonar outras crenças que impliquem em α.
3. **Revisão ($K * \alpha$):** Uma informação (crença) α é acrescentada ao conjunto K e para manter a consistência pode ser necessário abandonar outras crenças de K.

Descreveremos a seguir: a linguagem que usaremos para representar cada crença; algumas representações possíveis para um conjunto de crenças; os principais postulados que fundamentam a teoria de revisão de crenças; e duas relações importantes.

2.1 Representação

Primeiramente precisamos saber como representar cada *estado epistêmico* (de conhecimento, crença ou informação), sendo que usualmente é utilizado uma representação proposicional. Desta forma, iremos utilizar a linguagem L, que compreende a linguagem proposicional e seus conectivos lógicos. Usaremos a notação $K \vdash \alpha$, se K implica logicamente α, $K \nvdash \alpha$, se K não implica logicamente α, e *Cn(K)* para o conjunto de todas as conseqüências lógicas de K.

Em seguida veremos algumas das representações mais comuns utilizadas para representar estados epistêmicos:

- **Conjunto de Crenças:** É um conjunto de fórmulas logicamente fechado, ou seja, se $K \vdash \alpha$, sendo K um conjunto, então $\alpha \in K$. Ao utilizar tal representação, estamos propensos a trabalhar com conjuntos infinitos, o que não é apropriado do ponto de vista computacional.

- **Base de Crenças:** Supõe que um conjunto de crenças possui uma base para inferir todas as crenças de um conjunto. B_k é uma base para um conjunto de crenças K se e somente se B_k é um subconjunto finito de K e o fecho lógico de B_k é igual ao conjunto K. Logicamente podemos ter várias bases para um mesmo conjunto K. Do ponto de vista computacional, por trabalhar com conjuntos finitos, essa representação tem sido bastante utilizada. É a representação utilizada por [4] e a qual seguiremos também.

- **Mundos Possíveis:** Baseia-se na idéia de um conjunto W_k de mundos possíveis. Qualquer conjunto de crenças K pode ser representado pelo subconjunto W_k dos mundos possíveis em que todas as sentenças de K são verdadeiras. Essa representação é muito utilizada por lógicos e filósofos e começa a ser usada computacionalmente.

2.2 Postulados

Vamos assumir que estamos utilizando a representação dos estados epistêmicos como no modelo de conjunto de crenças. A motivação por trás dos postulados é que o *Princípio da Mudança Mínima* seja satisfeito a cada operação.

2.2.1 Expansão

Dado um conjunto de crenças K e uma crença α, definimos a expansão como: $K + \alpha = Cn(K \cup \alpha)$.

2.2.2 Revisão

Dado um conjunto de crenças K e uma crença α, os seis postulados básicos da revisão (mais dois extras) são:

1. $K * \alpha$ é um conjunto de crenças;
2. $\alpha \in K * \alpha$;
3. $K * \alpha \subseteq K + \alpha$;
4. Se $\neg\alpha \notin K$, então $K + \alpha \subseteq K * \alpha$;
5. $K * \alpha$ é inconsistente sse $\vdash \neg\alpha$;
6. Se $\alpha \leftrightarrow \beta$, então $K * \alpha = K * \beta$;
7. $K * (\alpha \wedge \beta) \subseteq (K * \alpha) + \beta$;
8. Se $\neg\beta \notin K * \alpha$, então $(K * \alpha) + \beta \subseteq K * (\alpha \wedge \beta)$.

2.2.3 Contração

Dado um conjunto de crenças K e uma crença α, os seis postulados básicos da contração (mais dois extras) são:

1. $K - \alpha$ é um conjunto de crenças;
2. $K - \alpha \in \alpha$;
3. Se $\alpha \notin K$, então $K - \alpha = K$;

4. $\nvdash \alpha$, então $\alpha \notin K - \alpha$;

5. $K \subseteq (K - \alpha) + \alpha$;

6. Se $\vdash \alpha \leftrightarrow \beta$, então $K - \alpha = K - \beta$;

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

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

2.3 Revisão versus Contração

Existem duas relações importante entre os operadores de revisão e contração, conhecidas como:

- **Identidade de Levi:** $K * \alpha = (K - \neg\alpha) + \alpha$
- **Identidade de Harper:** $K - \alpha = K \cap (K * \neg\alpha)$

3 Lógica temporal - CTL

Para representar as informações em nosso sistema iremos utilizar lógica temporal. Lógicas temporais são um tipo de lógica onde se pode representar proposições relacionadas com o tempo. Por exemplo, podemos representar informações como: "Eu SEMPRE estou com fome"ou "Eu ficarei com fome ATÉ que eu coma". As duas principais lógicas temporais são a LTL (Linear Tree Logic) e a CTL (Computation Tree Logic) [3], a primeira itera através dos caminhos, enquanto que a segunda utiliza ramificações. Podemos diferenciá-las também através dos quantificadores: a segunda utiliza quantificadores sobre os caminhos, enquanto que na primeira o quantificador universal está implícito. Apesar de podermos expressar propriedades em LTL que não são possíveis em CTL, em sua maioria uma fórmula em LTL pode ser representado em CTL, portanto iremos utilizar a CTL como nossa linguagem de representação.

A idéia por trás desta lógica é quantificar as possíveis execuções

de um programa através da noção de caminhos que existem num espaço de estados do sistema. A sintaxe da CTL é dada pela seguinte definição:

$$\phi ::= p|\neg\phi|\phi \wedge \phi|(AX\phi)|(AG\phi)|(AF\phi)|$$
$$(EX\phi)|(EG\phi)|(EF\phi)|E(\phi \cup \phi)|A(\phi \cup \phi)$$

onde p é um átomo proposicional, $\neg$ e $\wedge$ são os operadores lógicos usuais e os demais são operadores de tempo. Cada operador temporal é composto por um quantificador de caminho (E, "existe um caminho", ou A, "para todo caminho") seguido por um quantificador de estado (X, "próximo estado no caminho", U, "até", G, "globalmente", ou F, "futuramente").

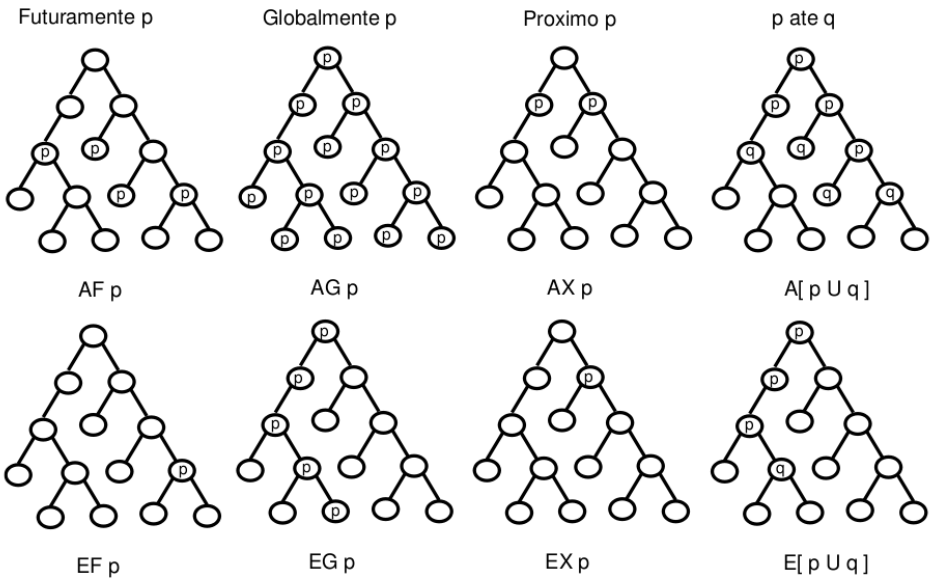

Figura 1: Operadores Temporais da CTL

Semântica CTL: Seja M uma estrutura de Kripke e $\pi(i)$ o i-ésimo estado s_i de um caminho, então dizemos que se $M, s \models \phi$, então ϕ é verdadeiro no estado s de M. Assim temos:

1. $M, s \models p$ *sse* $p \in L(s)$
2. $M, s \models \neg\phi$ *sse* $M, s \not\models \phi$
3. $M, s \models \phi_1 \wedge \phi_2$ *sse* $M, s \models \phi_1$ *e* $M, s \models \phi_2$
4. $M, s \models EX\phi$ *sse existe um estado* s' *de* M
   tal que $(s, s') \in R$ *e* $M, s' \models \phi$
5. $M, s \models EG\phi$ *sse existe um caminho* π *de* M
   tal que $\pi(1) = s$ *e* $\forall i \ \geq \ 1 \bullet M, \phi(i) \models \phi$
6. $M, s \models E(\phi_1 U \phi_2)$ *sse existe um caminho* π *de* M
   tal que $\pi(1) = s$ *e* $\exists i \ \geq 1 \bullet$
   $(M, \pi(i) \models \phi_2 \wedge \ \forall j, i > j \leq 1 \bullet M, \pi(j) \models \phi_1)$

Ao trabalharmos com CTL iremos apenas utilizar a combinação de quantificadores EX, EG e E(ϕUϕ), afim de simplificarmos os algoritmos de verificação. Os demais operadores podem ser derivados através das fórmulas:

$$AX\phi = \neg EX\neg\phi$$
$$AG\phi = \neg EF\neg\phi$$
$$AF\phi = \neg EG\neg\phi$$
$$EF\phi = E[true \cup \phi]$$
$$A[\phi \cup \beta] = \neg E[\neg\beta \cup \neg\phi \wedge \neg\beta] \wedge \neg EG\neg\beta$$

4 Verificação de modelos

A verificação de modelos é uma das técnicas mais conhecidas de verificação de sistemas que podem ser modelados como uma máquina de estados finitos, e vem sendo desenvolvida desde a década de 80. Basicamente consiste em: dado uma propriedade em alguma lógica temporal, verificar se a máquina de estados finitos que representa o sistema satisfaz aquela fórmula. Ou seja, verificar se uma fórmula f é satisfeita por um grafo G de estados.

Para representar esse grafo de estados, iremos utilizar uma *Estrutura de Kripke*. Uma estrutura de Kripke é definida como uma tupla (S, R, I, L), onde S é um conjunto de estados, R é o conjunto das transições, I é o conjunto dos estados iniciais e L é uma função que leva cada estado ao conjunto de proposições que são verdadeiras nele. Para modelar um estado em *deadlock*, basta criar uma transição do estado para si mesmo. A figura a seguir representa uma estrutura de Kripke onde: P={a, b, c}, S={s0, s1, s2}, R={(s0, s1), (s0, s2), (s1, s0), (s1,s2), (s2,s2)}, I={s0} com L(s0)={a,b}, L(s1)={b,c} e L(s2)={c}.

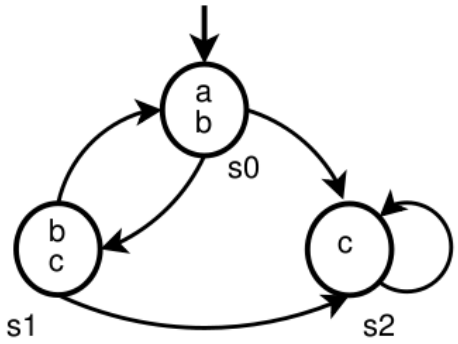

Figura 2: Estrutura de Kripke

No entanto essa representação por meio de um grafo de estados sofre de um sério problema, o *Problema de Explosão de Estados*, ou seja, conforme a complexidade do sistema aumenta, o número de estados cresce exponencialmente tornando a verificação da estrutura inviável devido ao armazenamento de estados. Portanto outras abordagens foram desenvolvidas para tentar solucionar o problema da explosão de estados. As mais conhecidas são: *Verificação de Modelos Simbólica* e *Verificação de Modelos Limitada*.

Iremos utilizar a abordagem de verificação de modelos limitada [2], ou *bounded model checking* (a partir de agora nos referiremos a ela como BMC), em nosso trabalho.

4.1 Verificação de modelos limitada

A solução proposta pela BMC é de colocar um limite k no tamanho máximo dos caminhos. A partir desse limite é possível traduzir a estrutura de Kripke e a fórmula em CTL para fórmulas proposicionais, verificando sua satisfazibilidade num resolvedor SAT. Para podermos executar a tradução precisamos de uma nova semântica para a CTL, que chamaremos *Semântica CTL k-limitada* e para o operador EG é necessário uma construção chamada k-loop , isto é, um caminho de tamanho máximo k contendo um loop.

A semântica CTL k-limitada, $M, s \models^k \phi$, $k \geq 0$, é definida por:

1. $M, s \models^k p$ *sse* $p \in L(s)$
2. $M, s \models^k \neg\phi$ *sse* $M, s \not\models^k \phi$
3. $M, s \models^k \phi_1 \wedge \phi_2$ *sse* $M, s \models^k \phi_1$ *e* $M, s \models^k \phi_2$
4. $M, s \models^k EX\phi$ *sse existe um estado* s' *de* M
   tal que $(s, s') \in R$ *e* $M, s' \models^{k-1} \phi$
5. $M, s \models^k EG\phi$ *sse existe um* $k-loop$ π *de* M
   tal que $\pi(1) = s$ *e* $\forall i \ \geq 1 \bullet M, \phi(i) \models^{k-i} \phi$
6. $M, s \models^k E(\phi_1 U \phi_2)$ *sse existe um caminho* π *de* M
   tal que $\pi(1) = s$ *e* $\exists i \geq 1 \bullet$
   $(M, \pi(i) \models^{k-i} \phi_2 \wedge \forall j, i > j \leq 1 \bullet M, \ \pi(j) \models^{k-j} \phi_1)$

Desta forma, com um limite k, traduzimos uma estrutura de Kripke K em uma fórmula proposicional K^k e uma fórmula em CTL ϕ numa fórmula clássica A_ϕ^k e podemos submeter a fórmula $K^k \wedge A_\phi^k$ no resolvedor SAT para obter um veredito. Se a fórmula é insatisfazível, isso significa que $\neg\phi$ vale na estrutura de Kripke K até o limite estabelecido. Caso contrário uma valoração v é retornada pelo resolvedor SAT, a qual representa um caminho através do modelo no qual ϕ vale.

5 Algoritmo

Até aqui, discorremos sobre o processo de revisão de crenças, suas características e postulados. Vimos também uma forma de modelar os estados epistêmicos através de uma lógica temporal conhecida como CTL. E por fim analisamos o processo de verificação de modelos, a fim de encontrarmos um modo de verificar o conjunto de crenças atrás de inconsistências e apontá-las. Podemos ver facilmente que o processo de revisão de crenças está altamente ligado tanto à CTL, quanto à verificação de modelos.

Figura 3: Verificação de Modelos com Revisão de Crenças

Em [5] foi mostrado que os postulados AGM podem ser aplicados somente para lógicas que são decomponíveis, o que ocorre com CTL. No entanto, o uso de conjuntos logicamente fechados leva a problemas do ponto de vista computacional, já que são tipicamente infinitos.

Para bases de crenças, [7] mostrou que se uma lógica é compacta e monotônica, as construções típicas podem ser aplicadas. Mas CTL e outras lógicas baseadas em tempo discreto não são compactas. No entanto, em [4] foi mostrado que utilizando BMC e uma tradução, tanto da especificação quanto das fórmulas em CTL, para lógica proposicional, de tal forma que um resolvedor SAT possa ser aplicado, a existência de um modelo revisado é garantido e facilmente traduzido para o nível da especificação.

Nosso trabalho segue a linha proposta em [4]. Iremos utilizar CTL para representar nossas crenças e usar o método de Bounded Model

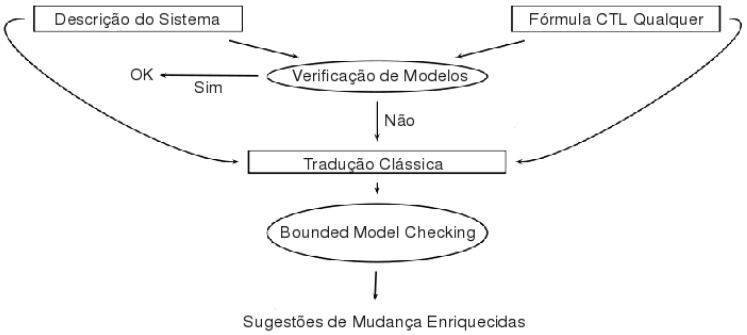

Figura 4: Revisão de Crenças com Bounded Model Checking

Checking (BMC) a fim de encontrarmos inconsistências e corrigi-las, até conseguir um conjunto consistente, retornando sugestões de alterações para o usuário. A idéia do algoritmo é receber uma especificação, transcrevê-la para a lógica proposicional clássica e por fim aplicar o processo de revisão por BMC, gerando as sugestões de mudanças.

Basicamente, traduzimos a estrutura de Kripke K e a fórmula CTL ϕ para lógica clássica, gerando K^k e A^k_ϕ. Em seguida submetemos a fórmula $K^k \wedge A^k_\phi$ num resolvedor SAT. A partir da resposta, geramos as sugestões de mudança no modelo, criando uma fórmula proposicional K_{rev}, representando o modelo revisado.

Basta agora traduzir K_{rev}, uma fórmula proposicional clássica, de volta em sugestões de alteração na especificação original K. Para executar a tarefa de tradução, temos que comparar a tradução inicial K^k com a teoria revisada K_{rev} e trazer as diferenças de volta para a especificação. Feita a comparação, basta interpretar a especificação revisada.

Desta forma o algoritmo pode ser resumido nos passos a seguir, representado na figura 5:

1. **Entrada:** A entrada do algoritmo será o modelo representado por uma estrutura de Kripke e uma fórmula em ACTL (fragmento universal da CTL);

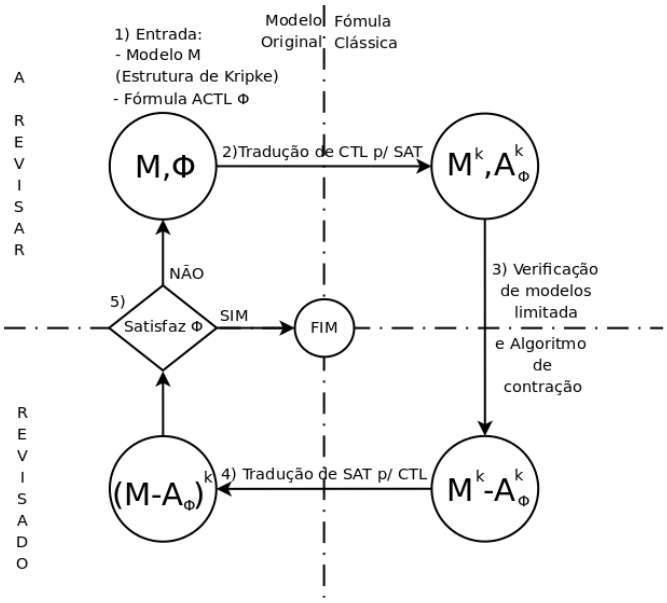

Figura 5: Representação do algoritmo de revisão de crenças com BMC

2. **Tradução inicial:** A tradução do modelo e da fórmula para lógica proposicional;
3. **Verificação inicial e Contração:** Verificação da fórmula traduzida e contração da fórmula;
4. **Tradução reversa:** A tradução do modelo com as sugestões de volta para a linguagem original;
5. **Verificação final:** Verificação se o modelo satisfaz a fórmula original.

Esses passos serão descritos mais detalhadamente a seguir.

5.1 Entrada

A entrada do algoritmo será formada pelo modelo numa estrutura de Kripke, como descrito anteriormente. E a fórmula a ser verificada deve ser descrita em ACTL, o fragmento universal da CTL, que consiste das fórmulas temporais na forma: $AG\alpha$, $AF\alpha$, $AX\alpha$ e $A(\alpha R\beta)$, e a negação somente aplicada a átomos proposicionais.

5.2 Tradução inicial

A tradução para lógica proposicional é uma implementação do algoritmo proposto por [6] para uma estrutura de Kripke. Foi utilizado uma implementação para ECTL, um fragmento da CTL, onde a negação só pode ser aplicada a proposições, seguindo a semântica CTL k-limitada demonstrada anteriormente. Como queremos revisar em relação a fórmulas em ACTL, que pode ser definida como $\{\neg\varphi | \varphi \in ECTL\}$, iremos verificar sua negação, ou seja, uma fórmula em ECTL. E como pretendemos utilizar um resolvedor SAT para verificar a satisfazibilidade da fórmula, precisamos gerá-la em CNF, facilitando os aspectos computacionais.

Seja $K = (S, R, I, L)$ uma estrutura de Kripke, seja $M = (W, T, \iota, \nu)$ um modelo definido como $M = S$, $T = R$, $\iota = I$ (iremos considerar que a estrutura contêm apenas um estado inicial) e $\nu = L$, e seja $k \in \mathbb{N}^+$. O $k - modelo$ para M é uma estrutura $M_k = ((W, Caminho_k, \iota), \nu)$ onde $Caminho_k$ é um conjunto de todos os $k = caminhos$ de M. E uma estrutura $M'_k = ((W', Caminho'_k, \iota), \nu')$ é um submodelo de M_k se $Caminho'_k \subseteq Caminho_k$, $W' = Estados(Caminho'_k)$ e $\nu' - \nu|_{W'}$.

Para cada estado de W representamos como um vetor de variáveis proposicionais $w = (w[1], ..., w[n])$ onde $n = \lceil log_2(|W|) \rceil$. E iremos precisar da função f_k, que determina o número de $k - caminhos$ de um submodelo M'_k suficientes para checar uma fórmula em ECTL, definida em [6] e listada abaixo:

- $f_k(p) = f_k(\neg p) = 0$, onde $p \in \mathcal{PV}$;
- $f_k(\alpha \vee \beta) = max\{f_k(\alpha),\ f_k(\beta)\}$;
- $f_k(\alpha \wedge \beta) = f_k(\alpha) + f_k(\beta)$;
- $f_k(EX\alpha) = f_k(\alpha) + 1$;
- $f_k(EG\alpha) = (k+1) * f_k(\alpha) + 1$;
- $f_k(E(\alpha U \beta)) = k * f_k(\alpha) + f_k(\beta) + 1$.

Definimos então as seguintes fórmulas proposicionais em CNF:

- $lit(0,p) = \neg p$ e $lit(1,p) = p$;
- $I_s(w) := \bigwedge_{l=1}^{n} lit(s[l], w[j])$;
- $T(w,v) := \begin{cases} \bigvee_{m=1}^{|T|} t_m \\ \bigwedge_{\substack{m=1 \\ (x,y)\in T}}^{|T|} \left[\begin{array}{c} \bigwedge_{l=1}^{n} (\neg t_m \vee lit(x[l], w[l])) \\ \bigwedge_{l=1}^{n} (\neg t_m \vee lit(y[l], v[l])) \\ \wedge (t_m \vee \bigvee_{l=1}^{n} lit(x[l], w[l]) \vee \bigvee_{l=1}^{n} lit(x[l], w[l])) \end{array} \right] \end{cases}$;
- $p(w) := V(w,p) := \begin{cases} (\bigvee_{x\,:\,p\,\in\,\nu(x)} s_i) \wedge \\ \bigvee_{x\,:\,p\,\in\,\nu(x)} D(s_i, (\bigwedge_{l=1}^{n} lit(x[l], w[l]) \wedge p)) \end{cases}$;
- $\neg p(w) := V(w, \neg p) := \begin{cases} (\bigvee_{x\,:\,p\,\notin\,\nu(x)} s_i) \wedge \\ \bigvee_{x\,:\,p\,\notin\,\nu(x)} D(s_i, (\bigwedge_{l=1}^{n} lit(x[l], w[l]) \wedge \neg p)) \end{cases}$;
- $H(w,v) := \bigwedge_{l=1}^{n} (\neg w[l] \vee v[l]) \wedge (w[l] \vee \neg v[l])$;
- $L_{k,j}(l) := T(w_{k,j}, w_{l,j})$ que codifica um loop no $k - caminho\ j$.

Iremos precisar também de uma fórmula que define uma variável proposicional para uma fórmula em CNF qualquer. Assim, seja $\alpha = v_1 \wedge ... \wedge v_m$, onde $v_1, ..., v_m$ são disjunções. Definimos a função D, que define a variável a para a fórmula α, da seguinte forma:

$$D(a, \alpha) := \bigwedge_{i=1}^{n} (\neg a \vee v_i) \wedge \bigwedge_{i=1}^{m} (a \vee v_{\neg i})$$

onde $v_{\neg 1}, ..., v_{\neg m}$ são as disjunções de $\neg\alpha$, supondo que $\neg\alpha$ esteja em CNF.

Definimos então a fórmula proposicional $[M^{\varphi,s}]_k$ como a tradução do modelo para lógica clássica:

$$[M^{\varphi,s}]_k := I_s(w_{0,0}) \wedge \bigwedge_{j\in LL\varphi} \bigwedge_{i=0}^{k} T(w_{i,j}, w_{i+1,j})$$

onde $w_{0,0}$ e $w_{i,j}$ para $i = 0, ..., k$ e $j \in LL\varphi$ representam os estados através dos $k - caminhos$, e $|LL\varphi| = f_k(\varphi)$.

O próximo passo é traduzir a fórmula ECTL φ para uma fórmula proposicional. Usaremos $[\varphi]_k^{[m,n]}$ para denotar a tradução de uma fórmula ECTL φ em $w_{m,n}$ para uma fórmula proposicional. Definimos a seguir a tradução:

- $[p]_k^{[m,n]} := p(w_{m,n})$;
- $[\neg p]_k^{[m,n]} := \neg p(w_{m,n})$;
- $[\alpha \wedge \beta]_k^{[m,n]} := [\alpha]_k^{[m,n]} \wedge [\beta]_k^{[m,n]}$;
- $[\alpha \vee \beta]_k^{[m,n]} := \alpha_k^{m,n} \vee \beta_k^{m,n} \wedge D(\alpha_k^{m,n}, [\alpha]_k^{[m,n]}) \wedge D(\beta_k^{m,n}, [\beta]_k^{[m,n]})$;
- $[EX\ \alpha]_k^{[m,n]} := \begin{cases} (\bigvee\limits_{i \in LL\varphi} ex_i^{m,n}) \wedge \\ \bigwedge\limits_{i \in LL\varphi} D(ex_i^{m,n}, H(w_{m,n}, w_{0,i}) \wedge [\alpha]_k^{[1,j]}) \end{cases}$;
- $[EG\ \alpha]_k^{[m,n]} := \begin{cases} (\bigvee\limits_{i \in LL\varphi} eg_i^{m,n}) \wedge \\ \bigwedge\limits_{i \in LL\varphi} D(eg_i^{m,n}, H(w_{m,n}, w_{0,i}) \wedge (\bigvee\limits_{l=0}^{k} L_{k,i}(l)) \wedge \bigwedge\limits_{j=0}^{k} [\alpha]_k^{[j,ij]}) \end{cases}$;
- $[E(\alpha U \beta)]_k^{[m,n]} := (\bigvee\limits_{i \in LL\varphi} eu_i^{m,n}) \wedge \bigwedge\limits_{i \in LL\varphi} \begin{bmatrix} \bigwedge\limits_{j=0}^{k} D(\alpha\beta_k^{j,i}, [\beta]_k^{[j,i]} \wedge \bigwedge\limits_{t=0}^{j-1} [\alpha]_k^{[t,i]}) \wedge \\ D(eu_i^{m,n}, H(w_{m,n}, w_{0,i}) \wedge \bigvee\limits_{j=0}^{k} \alpha\beta_k^{j,i}) \end{bmatrix}$;

Denotemos $[\varphi]_k^{[0,0]}$ como $[\varphi]_{M_k}$.

5.3 Verificação inicial e Contração

Tendo em mãos a tradução da estrutura de Kripke e da fórmula ECTL, precisamos rodar $[M^{\varphi,s}]_k \wedge [\varphi]_{M_k}$ num resolvedor SAT.

Ao verificar uma fórmula ϕ em um modelo, utilizando BMC, queremos saber se a propriedade $\neg\phi$ vale naquele modelo. De fato, se K e ϕ são inconsistentes, significa que K implica $\neg\phi$. Então se $[M^{\varphi,s}]_k \wedge [\varphi]_{M_k}$ é inconsistente, nada precisa ser feito com relação à revisão, e esse é o nosso caso ideal. Caso contrário, o resolvedor SAT irá gerar uma

valoração v que representa um caminho através de M que satisfaz a fórmula φ.

Figura 6: Bounded Model Checking requisitando Revisão de Crenças

A contração é feita ao transformar a valoração v em mudanças no modelo, gerando uma fórmula proposicional $A_{\neg v}$. Entretanto pode ser o caso em que $[M^{\varphi,s}]_k \wedge [\varphi]_{M_k} \wedge A_{\neg v}$ seja uma fórmula satisfeita por uma valoração v_2. Assim precisamos iterar m vezes até que alcancemos uma fórmula K_{rev} inconsistente.

$$K_{rev} = K^k \wedge A^k_\phi \wedge A_{\neg v} \wedge \cdots \wedge A_{\neg v_m}$$

Como o número de valorações é finito, o processo sempre termina.

5.4 Tradução reversa

Durante o processo de verificação e contração, iremos encontrar valorações que representam um caminho no modelo que satisfaz φ, o qual devemos modificar para que falsifiquemos a fórmula. Para tal precisamos observar as seguintes variáveis na tradução que representam o caminho:

- As **variáveis de estado** $w_{i,j}$ (ou seja todo o seu vetor), que representa o estado i do caminho j;
- As **variáveis de transição** $t_m^{i,i+1,j}$, que representa uma transição entre os estados $i, i+1$ do caminho j;
- As **propriedade** $p_{i,j}$, que representa seu valor no estado i do caminho j.

Essas variáveis identificam os diversos caminhos gerados pela CTL, no entanto, com uma valoração deles, podemos identificar seus representantes no modelo original. Assim, a partir da valoração v, obser-

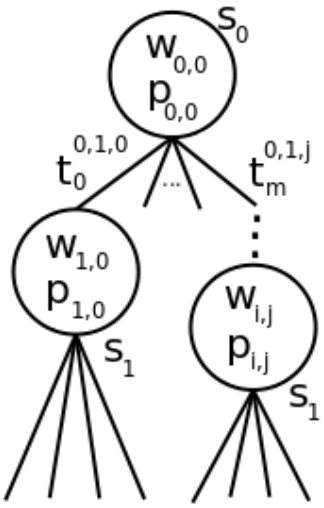

Figura 7: Variáveis utilizadas para gerar as sugestões

vando as variáveis descritas acima, podemos reconstruir os caminhos dentro do modelo original. Portanto basta escolhermos uma das variáveis para modificar seu representante no modelo original e acrescentarmos essa mudança na fórmula proposicional para rodarmos novamente a verificação. Essas modificações no modelo podem ser feitas das seguintes formas, com suas respectivas fórmulas proposicionais:

- **Remover um estado:**
  $R_s(v) = \bigwedge_{j \in LL\varphi} \bigwedge_{i=0}^{k} \bigvee_{l=0}^{n} \neg lit(v[l], w_{i,j}[l])$,
  onde v é a valoração do vetor do estado a ser excluído;

- **Remover uma transição:**
  $R_t(v) = \bigwedge_{j \in LL\varphi} \bigwedge_{i=0}^{k-1} \left[\bigvee_{l=0}^{n} \neg lit(x[l], w_{i,j}[l]) \vee \bigvee_{l=0}^{n} \neg lit(y[l], w_{i+1,j}[l]) \vee \neg t_m^{i,i+1,j} \right]$,
  onde (x, y) formam a transição $t_m^{i,i+1,j}$ definida por v;

- **Modificar uma propriedade:**
  $R_p(v) = \bigwedge_{j \in LL\varphi} \bigwedge_{i=0}^{k} \left[\bigvee_{l=0}^{n} \neg lit(v[l], w_{i,j}[l])) \vee \neg p_{i,j} \right]$,
  onde v é a valoração do estado onde p deve ser modificada.

Ao obtermos uma valoração, todas estas opções são possíveis, portanto cabe ao usuário decidir qual a melhor decisão a tomar, lembrando

que o objetivo da revisão de crenças é executar a menor quantidade de mudanças possíveis.

5.5 Verificação final

Ao término do algoritmo, temos que a especificação foi revisada e a fórmula ACTL ψ vale na estrutura de Kripke *até o limite k estabelecido*. No entanto pode ser que a fórmula ainda não seja satisfeita no modelo por um caminho de tamanho maior que k. Neste caso precisamos rodar o algoritmo novamente com um limite k maior, até que a fórmula seja satisfeita no modelo como um todo.

Algumas abordagens podem ser adotadas para tratar a escolha do k, procurando encontrar o limite ideal:

- Podemos começar com um k baixo e iterar até atingir o objetivo;
- Ou podemos utilizar um verificador de modelos comum para encontrar um caminho que falsifica a fórmula, definindo k como o tamanho desse caminho.

6 Exemplo

Para deixar mais claro a execução do método, nos utilizaremos de um simples exemplo. Seja $M = (S, R, \iota, \nu)$ o modelo representado pela estrutura de Kripke abaixo, e $\psi = AG\ a$. Queremos revisar M a respeito de ψ e portanto verificar se $M \models AG\ a$.

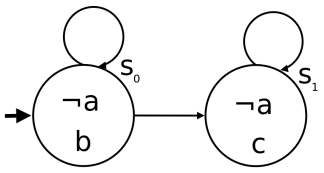

Figura 8: Estrutura de Kripke para o exemplo

Definimos nossa estrutura de Kripke da seguinte forma: $\iota = \{s_0\}$, $W = \{s_0, s_1\} = \{0,\ 1\}$, $T = \{(s0, s0),\ (s0, s1),\ (s1, s1)\}$, $\nu(s_0) = \{b\}$,

$\nu(s_1) = \{c\}$ e $\varphi = \neg\psi = E(T\ U\ \neg\ a)$ como a negação da fórmula que queremos revisar.

Iremos adotar $k = 1$, pois isso simplifica nossas fórmulas e também podemos ver que é suficiente um caminho de tamanho 1 para verificar a fórmula.

Calculando $f_k(\varphi)$, lembrando que $|LL\varphi| = f_k(\varphi)$, temos:

$$f_1(E(T\ U\ \neg\ a))\ =\ 1 * f_1(T)\ + f_1(\neg a)\ +\ 1\ =\ 1$$

As fórmulas para a tradução do modelo ficam da seguinte forma:

$$I_s(w_{0,0})\ =\ \neg w_{0,0}$$

$$T(w_{0,0}, w_{1,0})\ =\ \begin{cases} t_0 \vee t_1 \vee t_2\ \wedge \\ \neg t_0\ \vee\ \neg w_{0,0}\ \wedge \\ \neg t_0\ \vee\ \neg w_{1,0}\ \wedge \\ t_0\ \vee\ w_{0,0}\ \vee\ w_{1,0}\ \wedge \\ \neg t_1\ \vee\ \neg w_{0,0}\ \wedge \\ \neg t_1\ \vee\ w_{1,0}\ \wedge \\ t_1\ \vee\ w_{0,0}\ \vee\ \neg w_{1,0}\ \wedge \\ \neg t_2\ \vee\ w_{0,0}\ \wedge \\ \neg t_2\ \vee\ w_{1,0}\ \wedge \\ t_2\ \vee\ \neg w_{0,0}\ \vee\ \neg w_{1,0} \end{cases}$$

E a fórmula para a tradução da ECTL da seguinte forma:

$$[E(T\ U \neg a)]_1^{[0,0]}\ =\ \begin{cases} H(w_{0,0},\ w_{0,0})\ \wedge \\ (\alpha_{0,0}\ \vee\ \alpha_{1,0})\ \wedge \\ D(\alpha_{0,0},\ [\neg a]_1^{0,0})\ \wedge \\ D(\alpha_{1,0}, [\neg a]_1^{1,0}) \end{cases}$$

$$H(w_{0,0},\ w_{0,0}) = \begin{cases} \neg w_{0,0}\ \vee\ w_{0,0}\ \wedge \\ w_{0,0}\ \vee\ \neg w_{0,0} \end{cases}$$

$$D(\alpha_{0,0},\ [\neg a]_1^{0,0}) \ = \ \begin{cases} \neg\alpha_{0,0} \vee \neg w_{0,0} \vee \neg a_{0,0} \\ \neg\alpha_{0,0} \vee \neg w_{0,0} \vee \neg a_{0,0} \\ \neg\alpha_{0,0} \vee \neg a_{0,0} \\ \alpha_{0,0} \vee w_{0,0} \vee a_{0,0} \\ \alpha_{0,0} \vee \neg w_{0,0} \vee a_{0,0} \end{cases}$$

$$D(\alpha_{1,0},\ [\neg a]_1^{1,0}) \ = \ \begin{cases} \neg\alpha_{1,0} \vee \neg w_{1,0} \vee \neg a_{1,0} \\ \neg\alpha_{1,0} \vee \neg w_{1,0} \vee \neg a_{1,0} \\ \neg\alpha_{1,0} \vee \neg a_{1,0} \\ \alpha_{1,0} \vee w_{1,0} \vee a_{1,0} \\ \alpha_{1,0} \vee \neg w_{1,0} \vee a_{1,0} \end{cases}$$

Para rodarmos as fórmulas acima num resolvedor SAT, iremos renomeá-las da seguinte forma:

$$w_{0,0} \ = \ 1,\ w_{1,0} \ = \ 2,\ a_{0,0} \ = \ 3, a_{1,0} \ = \ 4,$$
$$t_0 \ = \ 5,\ t_1 \ = \ 6,\ t_2 \ = \ 7,\ \alpha_{0,0} \ = \ 8,\ \alpha_{1,0} \ = \ 9$$

Rodando a fórmula no resolvedor SAT temos a seguinte resposta: **-1 -2 -3 4 5 -6 -7 8 -9**. Que representa o caminho abaixo:

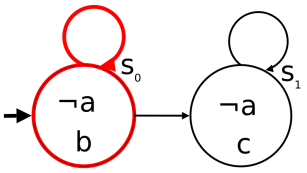

Figura 9: Primeira verificação

Como visto podemos escolher entre: remover um estado, remover uma transição ou modificar uma propriedade. Neste caso iremos simplesmente modificar uma propriedade, tornando a positivo no estado $s0$. Para isso iremos incluir a seguinte fórmula no modelo:

$$(w_{0,0} \vee a_{0,0}) \wedge (w_{1,0} \vee a_{1,0})$$

Com isso resultamos na seguinte estrutura de Kripke:

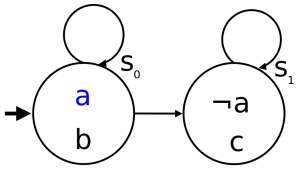

Figura 10: Resultado após a primeira mudança

Porém pode ser o caso de que a fórmula ainda não seja inconsistente, portanto rodamos o verificar SAT e resultamos numa nova valoração: **-1 2 3 -4 -5 6 -7 -8 9**. Que representa o caminho abaixo:

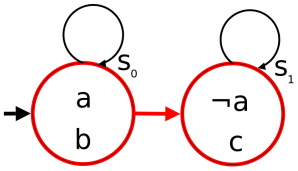

Figura 11: Segunda verificação

Novamente devemos escolher a mudança a ser feita. Desta vez escolheremos remover uma transição, excluindo a transição $(s0, s1)$. Para isso iremos incluir a seguinte fórmula no modelo:

$$(w_{0,0} \vee \neg w_{1,0} \vee \neg t_1)$$

Resultando na seguinte estrutura de Kripke:

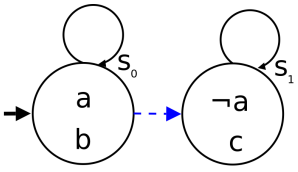

Figura 12: Remoção da transição

Talvez o resultado não tenha sido muito satisfatório, pois atingimos dois estados separados, portanto em vez de removermos uma transição, iremos apenas modificar uma propriedade, tornando a positivo no estado $s1$. Para isso incluiremos a seguinte fórmula no modelo:

$$(\neg w_{0,0} \vee a_{0,0}) \wedge (\neg w_{1,0} \vee a_{1,0})$$

Resultando na seguinte estrutura de Kripke:

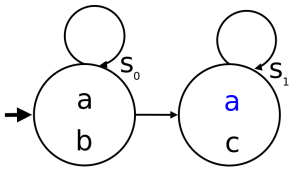

Figura 13: Inserção da propriedade a

Em ambos os casos chegamos a uma fórmula inconsistente e o processo de revisão de crenças chegou ao fim. Basta verificar se a estrutura revisada satisfaz a fórmula, o que podemos facilmente averiguar, concluindo assim a revisão por completo.

7 Conclusões e Resultados

Durante todo o trabalho conseguimos definir de forma satisfatória o processo de revisão de crenças com verificação de modelos limitada, sugerindo um algoritmo e uma implementação. Isso é um grande avanço, pois agora temos uma forma definida de como utilizar o método proposto por [4]. No entanto, será necessário implementar o algoritmo para que tenhamos uma melhor idéia de sua eficácia e validade. Portanto o próximo passo é implementar o algoritmo e executar testes a fim de analisar sua funcionalidade, dando abertura para melhoras e por fim chegando num produto final que cumpra o objetivo estabelecido, que esteja disponível para uso e consequentemente um veredito final sobre o método. Esta atividade será executada como início dos trabalhos de mestrado.

Referências

[1] C. E. Alchourròn, P. Gärdenfors, and D. Makinson. On the logic of theory change: Partial meet contraction and revision functions. *Journal of Symbolic Logic*, 50:510–530, 1985.

[2] Armin Biere, Alessandro Cimatti, Edmund Clarke, and Yunshan Zhu. Symbolic model checking without bdds. *Lecture Notes in Computer Science*, 1579:193–207, 1999.

[3] Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Logic of Programs, Workshop*, pages 52–71, London, UK, 1982. Springer-Verlag.

[4] Marcelo Finger and Renata Wassermann. Revising specifications with ctl properties using bounded model checking. In *SBIA '08: Proceedings of the 19th Brazilian Symposium on Artificial Intelligence*, pages 157–166, Berlin, Heidelberg, 2008. Springer-Verlag.

[5] Giorgos Flouris. On belief change in ontology evolution: Thesis. *AI Commun.*, 19(4):395–397, 2006.

[6] Wojciech Penczek, Bozena Wozna, and Andrzej Zbrzezny. Bounded model checking for the universal fragment of ctl. *Fundam. Inform.*, 51(1-2):135–156, 2002.

[7] Renata Wassermann and Sven Ove Hansson. Local change. In *In Fourth Symposium on Logical Formalizations of Common Sense Reasoning*, 1998.

A Parte Subjetiva

A.1 Desafios vencidos

Como todo trabalho acadêmico e de pesquisa, houve vários desafios durante todo o processo. Dentre eles, podemos destacar a vasta quantidade de material lido e estudado para adquirir um certo conhecimento do assunto, tanto de revisão de crenças, como de verificação de modelos e da linguagem CTL. Outro aspecto relacionado, foi a pesquisa por artigos e publicações que pudessem ajudar no desenvolvimento do algoritmo, resultando na descoberto de um trabalho que continha uma tradução.

Um dos maiores desafios vencidos foi o do entendimento da tradução e sua implementação para uma estrutura de Kripke. Essa etapa consumiu a maior parte do tempo do desenvolvimento do trabalho, sendo refeita várias vezes para que chegasse ao estado em que se encontra. Pessoalmente fico satisfeito em ver o resultado final e o desafio vencido, tendo em vista a dificuldade em executá-lo.

Por fim, um último desafio vencido foi, no que se refere à tradução e ao algoritmo, desenvolver o retorno da tradução e as sugestões no modelo original. Essa parte foi inteiramente original, pois não havia sido implementada nem desenvolvida. Apesar de ser baseada na tradução inicial e no texto base do algoritmo, a tradução reversa teve que ser completamente adaptada chegando no resultado encontrado na monografia.

A.2 Desafios a vencer

Alguns desafios tiveram que ser deixados para trabalhos futuros. E o principal deles é o da implementação do algoritmo. A princípio havíamos proposto implementá-lo, mas devido à dificuldade e à demora para atingir uma tradução satisfatória, decidimos por esperar

por maior maturidade teórica antes de sua implementação. Seguindo a mesma linha, havíamos proposto utilizar como entrada a linguagem formal SMV, que serve para descrever especificações. Pela praticidade e por causa do tempo, decidimos utilizar como entrada uma estrutura de Kripke, deixando a tradução de SMV para a estrutura de Kripke para trabalhos futuros. Esta parte já foi desenvolvida por outros projetos, como por exemplo o NuSMV (`http://nusmv.irst.itc.it/`), portanto pode ser usada futuramente.

E um último detalhe é que durante a pesquisa da tradução para lógica proposicional, nos deparamos com um artigo que sugere melhoras na tradução utilizada por nós. Porém como eram alterações pequenas e que não mudavam a estrutura básica da tradução, decidimos nos focar na tradução que já vinha sendo estudada e analisada, facilitando o trabalho e deixando as melhoras para uma próxima etapa.

A.3 Disciplinas relacionadas

A principal matéria relacionada ao trabalho foi a de *Métodos Formais*, que deu toda a fundamentação dos conceitos de lógica utilizados a todo momento no trabalho. Em seguida, temos as matérias de *Inteligência Artificial* e *Sistemas Baseados em Conhecimentos* que abordaram assuntos relacionados ao projeto, como as lógicas temporais e uso de ontologias, já que o projeto engloba tanto os temas de inteligência artificial, representação de conhecimento e linguagens formais.

A.4 Próximas etapas

Basicamente a próxima etapa é vencer os desafios citados anteriormente: melhora da tradução; implementação do algoritmo e; mudança da linguagem de entrada para SMV. Pretendo continuar este trabalho em estudos futuros e provavelmente num mestrado, onde como tarefa inicial pretendo completar as etapas citadas. O principal objetivo

futuro é tornar o algoritmo utilizável cotidianamente, gerando uma ferramenta capaz de gerar sugestões satisfatórias nas especificações.