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

Aluno: Bruno Vercelino da Hora **Orientador:** Marcelo Finger

Instituto de Matemática e Estatística - USP

Introdução

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, o cliente pode querer acrescentar ou modificar as funcionalidades ao sistema. Tudo isso requer que a especificação do software seja revista, mas isso é altamente custoso, tornando um processo automatizado necessário para diminuí-lo.

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.

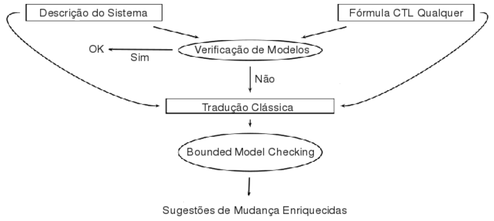


Figura 1: Processo de Revisão de Crenças com Verificação de Modelos Limitada

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.

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 consistente α, juntamente com suas conseqüências lógicas é adicionada ao conjunto. Ou seja: $K+\alpha = Cn(K \cup \alpha)$.
2. **Contração ($K-\alpha$):** A informação 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 α é acrescentada ao conjunto K e para manter a consistência pode ser necessário abandonar outras crenças de K.

Existem duas relações importante, que ficaram conhecidas como:

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

Lógica Temporal - CTL

Para representar as informações em nosso sistema iremos utilizar uma lógica temporal, a *Computation Tree Logic*, ou CTL. 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").

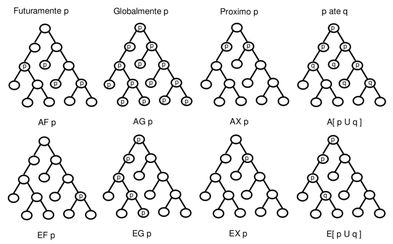
Ao trabalharmos com CTL iremos apenas utilizar a combinação de quantificadores EX, EG e E(ϕUϕ), afim de simplificarmos os algoritmos de verificação. Os demais operadores podem ser derivados através das fórmulas:

$AX\phi = \neg EX\neg\phi$ $\quad EF\phi = E[true \cup \phi]$

$AG\phi = \neg EF\neg\phi$ $\quad A[\phi \cup \beta] = \neg E[\neg\beta \cup \neg\phi \wedge \neg\beta] \wedge \neg EG\neg\beta$

$AF\phi = \neg EG\neg\phi$

Figura 2: Operadores Temporais da CTL

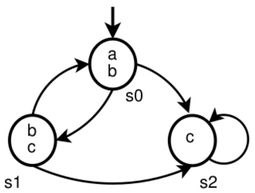


Verificação de modelos limitada

Verificação de modelos é uma técnica que consiste em verificar se um sistema, modelado como uma máquina de estados finitos, satisfaz uma determinada fórmula em alguma lógica temporal. Para representar esse grafo de estados, iremos utilizar uma *Estrutura de Kripke*.

Definição: 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.

Figura 3: Estrutura de Kripke



No entanto essa representação sofre do *Problema de Explosão de Estados*, ou seja, conforme a complexidade do sistema aumenta, o número de estados cresce exponencialmente. Iremos utilizar como solução a técnica de verificação de modelos limitada (*Bounded Model Checking*), que consiste em colocar um limite ***k*** no tamanho máximo dos caminhos.

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 de K no qual ϕ é válido.

Algoritmo

Nosso trabalho segue a linha proposta em [1]. O objetivo do algoritmo é receber uma estrutura de Kripke e uma fórmula em ACTL (fragmento universal da CTL), transcrevê-la para a lógica proposicional e por fim aplicar o processo de revisão por BMC. O algoritmo pode ser dividido da seguinte forma:

1. **Entrada:** A entrada do algoritmo será o modelo representado por uma estrutura de Kripke e uma fórmula em ACTL (fragmento universal da CTL);
2. **Tradução inicial:** A tradução do modelo e da fórmula para lógica proposicional;
3. **Verificação inicial:** Verificação da fórmula num resolvedor SAT;
4. **Contração:** Negação das possíveis valorações que satisfazem a a negação da fórmula a ser revisada;
5. **Volta da tradução:** A tradução do modelo com as sugestões de volta para a linguagem original;
6. **Verificação final:** Verificação se o modelo satisfaz a fórmula original.

A seguir descreveremos mais detalhadamente as etapas do algoritmo.

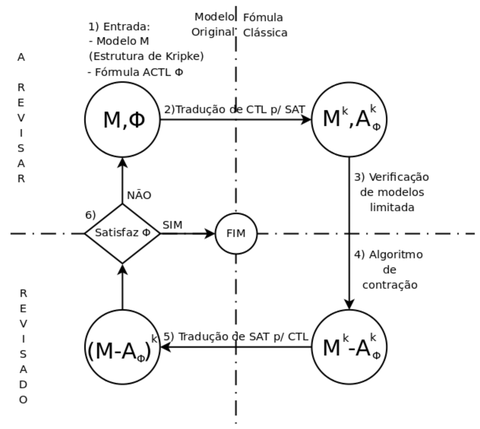


Figura 4: Algoritmo de revisão de crenças com BMC .

Tradução para lógica proposicional

A tradução para lógica proposicional é uma implementação do algoritmo proposto por [2] para uma estrutura de Kripke e como a fórmula será usada posteriormente em um resolvedor SAT, já estão em sua maioria em CNF. Foi utilizado uma implementação para ECTL, um fragmento da CTL, onde a negação só pode ser aplicada a proposições. No entanto como queremos revisar em relação a fórmulas em ACTL, que pode ser definida como $\{\neg\varphi \mid \varphi \in ECTL\}$, iremos verificar sua negação, ou seja, uma fórmula em ECTL.

Definições: 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 [2].

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.

Implementação da Tradução do Modelo

Definimos então as seguintes fórmulas propocisionais:

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

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 $|LL\varphi| = f_k(\varphi)$.

Implementação da Tradução da Fórmula

A 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:

- $[p]_k^{[m,n]} := p(w_{m,n})$;
- $[\neg p]_k^{[m,n]} := \neg p(w_{m,n})$;
- $[\alpha \wedge \beta]_k^{[m,n]} := [\alpha]_k^{[m,n]} \wedge [\beta]_k^{[m,n]}$;
- $[\alpha \vee \beta]_k^{[m,n]} := \alpha_k^{m,n} \vee \beta_k^{m,n} \wedge D(\alpha_k^{m,n}, [\alpha]_k^{[m,n]}) \wedge D(\beta_k^{m,n}, [\beta]_k^{[m,n]})$;
- $[EX\ \alpha]_k^{[m,n]} := (\bigvee_{i \in LL\varphi} ex_i^{m,n}) \wedge \bigwedge_{i \in LL\varphi} D(ex_i^{m,n}, H(w_{m,n}, w_{0,i}) \wedge [\alpha]_k^{[1,j]})$;
- $[EG\ \alpha]_k^{[m,n]} := (\bigvee_{i \in LL\varphi} eg_i^{m,n}) \wedge \bigwedge_{i \in LL\varphi} D(eg_i^{m,n}, H(w_{m,n}, w_{0,i}) \wedge (\bigvee_{l=0}^{k} L_{k,i}(l)) \wedge \bigwedge_{j=0}^{k} [\alpha]_k^{[j,j]})$;
- $[E(\alpha U \beta)]_k^{[m,n]} := (\bigvee_{i \in LL\varphi} eu_i^{m,n}) \wedge \bigwedge_{i \in LL\varphi} \left[\bigwedge_{j=0}^{k} D(\alpha\beta_k^{j,i}, [\beta]_k^{[j,i]} \wedge \bigwedge_{t=0}^{j-1} [\alpha]_k^{[t,i]}) \wedge\ D(eu_i^{m,n}, H(w_{m,n}, w_{0,i}) \wedge \bigvee_{j=0}^{k} \alpha\beta_k^{j,i}) \right]$;

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

Verificação e Contração

Após completada a tradução, tanto do modelo como da fórmula, para a linguagem clássica, checamos a fórmula $[M^{\varphi,s}]_k \wedge [\varphi]_{M_k}$ num resolvedor SAT. Caso a fórmula seja 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, a qual representa um caminho através do modelo no qual ϕ vale.

$$K^k \wedge A_\phi^k \longrightarrow \boxed{\text{SAT}} \xrightarrow{\text{Sim}} v$$

Figura 5: BMC requisitando Revisão de Crenças

A contração ocorre basicamente escolhendo uma alteração tendo em vista v, como descrito posteriormente, traduzindo essas mudanças numa fórmula $A_{\neg v}$ e verificando $[M^{\varphi,s}]_k \wedge [\varphi]_{M_k} \wedge A_{\neg v}$ novamente no resolvedor SAT. Entretanto pode ser o caso em que essa fórmula seja satisfeita por uma valoração v_2. Assim precisamos iterar m vezes até que alcancemos uma fórmula inconsistente. Como o número de valorações é finito, o processo sempre termina.

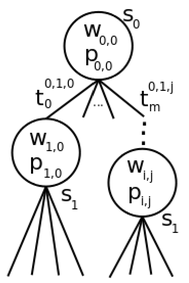
Sugestões no modelo original

As sugestões de modificação no modelo são encontradas paralelamente ao processo de contração. Para gerar as sugestões precisamos analisar as seguintes variáveis provenientes da tradução inicial:

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

Essas variáveis identificam os diversos caminhos da CTL, no entanto, com uma valoração deles podemos identificar seus representantes no modelo original. Assim, a partir da valoração v, observando 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, acrescentarmos essa mudança na fórmula proposicional e rodarmos novamente a verificação. Essas modificações no modelo podem ser feitas das seguintes formas, com suas respectivas fórmulas propocisionais:

Figura 6: Variáveis utilizadas para gerar as sugestões



- **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 mundanças possíveis.

Após iterarmos e atingirmos uma fórmula insatisfazível, chegamos ao término do algoritmo. Logo o modelo revisado será formado pelo modelo original e as mudanças escolhidas pelo usuário. No entanto obtemos um modelo revisado que satisfaz a fórmula até um *limite k*, e pode ser o caso em que o novo modelo ainda não satisfaça a fórmula por completo. Para tal é necessário averiguar sua satisfabilidade e se necessário executar o algoritmo novamente com um k maior.

Referências

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.

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

bruno.hora@usp.br