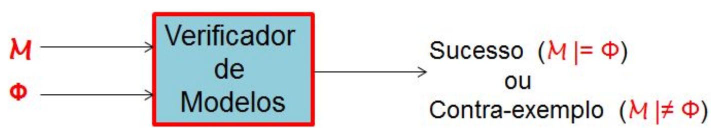
EP2

Verificação Simbólico de Modelos

Data de entrega: 10/12/2015

1. Objetivo: Implementar um Verificador Simbólico de Modelos CTL usando uma biblioteca de BDDs.

2. Introdução

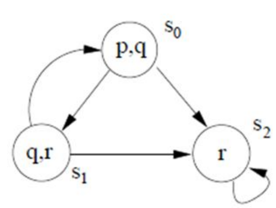


O problema da Verificação de Modelos é: dado M, s e Φ, queremos verificar se M, *s* |= Φ devolvendo SIM, se essa relação for verdadeira ou NÂO, caso contrário, isto é, M, *s* |≠ Φ. Porém, alternativamente podemos resolver o seguinte problema:

"Dado um modelo M e uma fórmula CTL Φ, encontre o conjunto A de todos os estados que satisfazem Φ",

e em seguida, basta verificar se $s \in A$.

A Figura 1 mostra um exemplo de modelo M , dado pela tupla $(S, \rightarrow, L)$ (estrutura de Kripke). É importante observar que para todo $s \in S$ existe um $s' \in S$ tal que (s,s') pertence a $\rightarrow$, isto é, existe uma transição a partir de todo estado $s \in S$.



$S = \{s_0, s_1, s_2\}$
$\rightarrow = \{(s_0,s_1), (s_0,s_2), (s_1,s_0), (s_1,s_2), (s_2,s_2)\}$
$L(s_0) = \{p,q\}$
$L(s_1) = \{q,r\}$
$L(s_2) = \{r\}$

Figura 1

3. Algoritmos de verificação de modelos

Os 4 algoritmos de verificação de modelos vistos em sala de aula (Apêndice A): SAT, SAT_{AF}, SAT_{EX} e SAT_{EU}, manipulam subconjuntos de estados de S e o modelo de transições de estados, dado pelas duplas $s \rightarrow s'$, com $s, s' \in S$. Note ainda que os 3 algoritmos, SAT_{AF}, SAT_{EX} e SAT_{EU}, realizam operações de pré-imagem para "subir" na árvore computacional. Para as fórmulas temporais temos 2 tipos de operações de pré-imagem (vide seção 6.3 do livro).

4.1 Pré-imagem Fraca ($Pre_{\exists}$)

A pré-imagem fraca de um conjunto de estados X, denotada por $Pre_{\exists}(X)$, devolve o conjunto de todos os estados $s \in S$ tal que, na estrutura de Kripke (modelo), existe uma aresta que sai de s e vai para algum estado $s' \in X$, ou seja:

$$Pre_{\exists}(X) = \{s \in S \mid \exists s', s \rightarrow s' \text{ e } s' \in X\} \qquad (1)$$

A $Pre_{\exists}(X)$ é usada nos algoritmos SAT_{EX} e SAT_{EU}.

4.2 Pré-imagem Forte ($Pre_{\forall}$)

A pré-imagem forte de um conjunto de estados X, denotada por $Pre_{\forall}(X)$, devolve o conjunto de todos os estados $s \in S$ tal que, na estrutura de Kripke (modelo), para todo $s' \in S$ que exista uma aresta que sai de s e leve a s' então $s' \in X$, ou seja:

$$Pre_{\forall}(X) = \{s \in S \mid \forall s', s \rightarrow s' \text{ implica que } s' \in X\}$$

(2)

A $Pre_{\exists}(X)$ é usada no algoritmo SAT_{AF}. Note que podemos computar a pré-imagem forte usando a pré-imagem fraca, da seguinte forma:

$$Pre_{\forall}(X) = S \setminus Pre_{\exists}(S-X) \qquad (3)$$

Assim, precisamos apenas implementar $Pre_{\exists}(X)$, e computar a pré-imagem forte em função desta.

5. Algoritmos de verificação de modelos simbólica

Para implementarmos, usando a biblioteca de BDDs, as versões simbólicas dos 4 algoritmos, SAT, SAT_{AF}, SAT_{EX} e SAT_{EU}, vamos especificá-los em termos de operações entre expressões lógicas, ao invés de operações entre conjuntos de estados. Para isso, precisaremos representar conjuntos de estados e o modelo (dado pelas relações $s \rightarrow s'$) em termos de expressões lógicas, codificadas como BDDs. Também precisaremos definir as operações entre conjuntos, incluindo a operação de pré-imagem fraca, em termos de operações entre expressões lógicas. Isso será visto em detalhes nessa seção.

Vamos usar a seguinte convenção para as operações lógicas e de conjuntos, transformadas em operações booleanas:

$\wedge$ ou $\cap$	* (ou & na biblioteca de BDDs PyEDA)
$\vee$ ou $\cup$	+ (ou | na biblioteca de BDDs PyEDA)

5.1 Representação fatorada de um conjunto de estados

Como vimos, um estado s é rotulado por um conjunto de proposições L(s), isto é L(s)={x_1, x_2, …, x_m}, sendo *m* o número de proposições do modelo. Assim, representaremos um estado s como uma conjunção de x_i , se $x_i \in$ L(s), e $\bar{x}_i$, se $x_i \notin$ L(S). Por exemplo, sendo m=3 e L(s)= {x_1, x_3}, temos s dado pela fórmula lógica $x_1 \wedge \bar{x}_2 \wedge x_3$, representada pela função booleana::

$$B_s = x_1 * \bar{x}_2 * x_3 \quad (4)$$

Um conjunto de estados X é representado pela disjunção das fórmulas de cada estado, ou seja, $B_X = B_{s1} \vee B_{s2} \vee B_{s3}$, representado pela função booleana:

$$B_X = B_{s1} + B_{s2} + B_{s3} \quad (5)$$

5.2 Representação fatorada do modelo

Dado uma função de transição de estados $\rightarrow$, a representação fatorada de uma aresta $s \rightarrow s'$, é dada pela conjunção entre a fórmula de s (isto é, conjunção de variáveis x_i e $\bar{x}_i$) e a fórmula de s' (isto é, conjunção de variáveis x'_i e $\overline{x'}_i$) . A representação fatorada completa para a relação de transição, $B_\rightarrow$, é a disjunção das fórmulas de cada aresta.

5.3 Função de Pré-imagem (fraca) simbólica

Seja B_X a representação fatorada do conjunto de estados X e $B_{X'}$, a representação fatorada do conjunto X' (isto é, envolvendo variáveis x'_i e $\overline{x'}_i$). Seja ainda o conjunto $B_\rightarrow$ a representação fatorada do modelo. Para computarmos de forma simbólica a pré-imagem fraca de um conjunto de estados X (Seção 6.3.3 do livro), dada pela Equação (1):

$$Pre_{\exists}(X) = \{s \in S \mid \exists s' {}_{,} s \rightarrow s' \text{ e } s' \in X\},$$

temos que primeiro verificar quais são as relações $s \rightarrow s'$ que levam para estados em X, isto é X'. Para isso, basta fazer a conjunção $B_{X'} \wedge B_{\rightarrow}$ $(B_{X'} * B_{\rightarrow})$. Note que na fórmula resultante são eliminados os fatores das disjunções em $B_{\rightarrow}$ que geram uma inconsistência $(p \vee \neg p')$ com as variáveis em $B_{X'}$, uma vez que $\perp \vee \varphi \equiv \varphi$.
FInalmente, usamos a operação Exists (vista na aula sobre BDDs) para "eliminar" **todas** as variáveis "linha" (Equação 6).

$$Pre_{\exists}(X) = \text{Exists } (x_i') \ (B_{X'} * B_{\rightarrow}) \text{ , com } i=1, \ldots, m \qquad (6)$$

Assim, todas as operações dos algoritmos SAT, SAT_{AF}, SAT_{EX} e SAT_{EU}, podem ser definidas de forma simbólica e podemos implementá-las usando a biblioteca de BDDs PyEDA.

6. Especificações da implementação

Neste EP, você deverá implementar um programa em Python, utilizando a biblioteca PyEDA, que recebe uma descrição de um modelo de transição M, uma fórmula CTL ϕ e um estado inicial s e decide se s satisfaz (ou não) a fórmula ϕ.

6.1. Representação do modelo

Iremos representar o conjunto de estados S com um número inteiro N, tal que |S|=N,, sendo que os estados de S são números 0, 1, … N.

A função de transição será descrita por pares de inteiros, por exemplo, (1,3) para $s_1 \rightarrow s_3$.

A função de rotulação L(s) será dada por uma lista de subconjuntos de [x_i], onde para cada estado i, L(i) é um subconjunto $\{x_j\}$.

Para ilustrar, o modelo descrito na Figura 1 é codificado da seguinte forma (substituindo p, q e r por x_1, x_2 e x_3):

```
3
[(0,1),(0,2),(1,0),(1,2),(2,2)]
[(x1,x2), (x2,x3) ,(x3) ]
```

6.2. Parser de fórmulas temporais

Fórmulas temporais CTL são fórmulas lógicas que utilizam os operadores básicos de Lógica Proposicional ($\vee$, $\wedge$, $\neg$) e operadores temporais EX, AX, EF, AF, EG, AG, EU e AU.

Para facilitar a leitura e manipulação das fórmulas temporais durante a verificação, iremos disponibilizar um parser (parser.py) para transformar fórmulas CTL (“strings de texto”) em sua árvore de análise (ParseTree) correspondente.

Neste Exercício-Programa, vamos utilizar uma notação simplificada para as fórmulas CTL:
As constantes verdadeiro ($\top$) e falso ($\bot$) serão representadas por 1 e 0, respectivamente.

Todos átomos serão da forma x_n, onde n é um número inteiro. Os operadores lógicos ($\vee$, $\wedge$, $\neg$) serão representados por (+, *, -) respectivamente.
Os operadores temporais são representados por pares de letras maiúsculas, sendo que a primeira pode ser A ou E e a segunda pode ser X, F, G ou U.

Além disso, usaremos a notação prefixa dos operadores binários, utilizando parênteses ao redor dos elementos, por exemplo: “+(ϕ_1)(ϕ_2)” ao invés de “$\phi_1 \wedge \phi_2$” e “EU(1)(ϕ_1)” ao invés de “E[$\top$ U ϕ_1]“. Os operadores unários são utilizados sem parentêses, por exemplo: “-x3”, “AX EG x2”

6.3. Exemplos de Teste

A entrada do seu programa é dada da seguinte forma (5 linhas):

```
N                                      // número de estados
[(n1,n2),(n3,n4), ...]                 // lista de pares, arestas do modelo de
Kripke
[(“x1”,”x2”, …),(“x2”, ... “x4”),...]   // lista com os rótulos dos N estados
AF EU(+(EG x1)(x2))(AF x1)                   // fórmula CTL a ser testada
k                                      // estado k de interesse
```

A saída de seu programa deve ter duas linhas: (1) uma lista de todos estados (número) que satisfazem a fórmula e (2) “*satisfaz*” ou “*não satisfaz*” se o estado de interesse satisfaz a fórmula ou não.

Use o exercício 6.12 (2) do livro para testar o seu programa para os dois modelos CTL da Figura 6.32.

Os modelos da Figura 6.32 seriam dados por: (note que a Figura 6.32(a) tem uma aresta incompleta, para este exercício, considere que o sentido é de s_3 para s_1.)

Modelo (a) e fórmula (a)

```
4
[(0,1),(0,2),(1,1),(1,2),(2,0),(2,1),(2,2),(3,0),(3,1),(3,3)]
[("x1","x2"),("x1"),(),("x2")]
AG +(x1)(-x2)
("x1")
```

Modelo (b) e fórmula (b)

```
3
[(0,2),(1,0),(1,1),(2,0),(2,1)]
[("x1"),("x2"),()]
EU(x2)(x1)
("x2")
```

Apêndice A. Pseudo-código dos algoritmos de verificação (não-fatorados)

function SAT(ϕ):
/* precondition: ϕ is an arbitrary CTL formula */
/* postcondition: SAT(ϕ) returns the set of states satisfying ϕ */
begin function
  case
    ϕ is $\top$: **return** S
    ϕ is $\bot$: **return** $\emptyset$
    ϕ is atomic formula: **return** $\{s \in S \mid \phi \in L(s)\}$
    ϕ is $\neg\phi_1$: **return** $S \setminus \mathrm{SAT}(\phi_1)$
    ϕ is $\phi_1 \wedge \phi_2$: **return** $\mathrm{SAT}(\phi_1) \cap \mathrm{SAT}(\phi_2)$
    ϕ is $\phi_1 \vee \phi_2$: **return** $\mathrm{SAT}(\phi_1) \cup \mathrm{SAT}(\phi_2)$
    ϕ is $\phi_1 \to \phi_2$: **return** $\mathrm{SAT}(\neg\phi_1 \vee \phi_2)$
    ϕ is $\mathrm{AX}\phi_1$: **return** $\mathrm{SAT}(\neg\mathrm{EX}\neg\phi_1)$
    ϕ is $\mathrm{EX}\phi_1$: **return** $\mathrm{SAT}_{EX}(\phi_1)$
    ϕ is $\mathrm{A}[\phi_1 \mathrm{U} \phi_2]$:
      return $\mathrm{SAT}(\neg(\mathrm{E}[\neg\phi_1 \mathrm{U}(\neg\phi_1 \wedge \neg\phi_2)] \vee \mathrm{EG}\neg\phi_2))$
    ϕ is $\mathrm{E}[\phi_1 \mathrm{U} \phi_2]$: **return** $\mathrm{SAT}_{EU}(\phi_1, \phi_2)$
    ϕ is $\mathrm{EF}\phi_1$: **return** $\mathrm{SAT}(\mathrm{E}[\top \mathrm{~U~} \phi_1])$
    ϕ is $\mathrm{EG}\phi_1$: **return** $\mathrm{SAT}(\neg\mathrm{AF}\neg\phi_1])$
    ϕ is $\mathrm{AF}\phi_1$: **return** $\mathrm{SAT}_{AF}(\phi_1)$
    ϕ is $\mathrm{AG}\phi_1$: **return** $\mathrm{SAT}(\neg\mathrm{EF}\neg\phi_1)$
  end case
end function

function $\mathrm{SAT}_{AF}(\phi)$:
/* pre: ϕ is an arbitrary CTL formula */
/* post: $\mathrm{SAT}_{AF}(\phi)$ returns the set of states satisfying $\mathrm{AF}\ \phi$ */
local var X, Y
begin
    $X := S$;
    $Y := \mathrm{SAT}(\phi)$;
    repeat until $X = Y$
        begin
            $X := Y$;
            $Y := Y \cup \{s \in S \mid$ for all s' with $s \to s'$ we have $s' \in Y\}$;
        end
    return Y
end

function $\mathrm{SAT}_{EU}(\phi, \psi)$:
/* pre: ϕ is an arbitrary CTL formula */
/* post: $\mathrm{SAT}_{EU}(\phi, \psi)$ returns the set of states satisfying $\mathrm{E}[\phi \mathrm{U} \psi]$ */
local var W, X, Y
begin
    $W := \mathrm{SAT}(\phi)$;
    $X := S$;
    $Y := \mathrm{SAT}(\psi)$;
    repeat until $X = Y$
        begin
            $X := Y$;
            $Y := Y \cup (W \cap \{s \in S \mid$ exists s' such that $s \to s'$ and $s' \in Y\}$;
        end
    return Y
end

function $\mathrm{SAT}_{EX}(\phi)$:
/* pre: ϕ is an arbitrary CTL formula */
/* post: $\mathrm{SAT}_{EX}(\phi)$ returns the set of states satisfying $\mathrm{EX}\ \phi$ */
local var X, Y
begin
    $X := \mathrm{SAT}(\phi)$;
    $Y := \{s_0 \in S \mid s_0 \to s_1$ for some $s_1 \in X\}$;
    return Y
end