Capítulo 4 : Verificação de Modelos baseada em SAT (1ª parte)

Introdução

O uso de relações de transição T na definição de um FOTS

$$\Sigma \equiv \langle X, I, T \rangle$$

é a forma mais flexível de exprimir o comportamento de um sistema dinâmico; este é o modelo dos *sistemas de transição de estados* com o qual se pode representar quase todos os sistemas dinâmicos *discretos*.

Existem sistemas dinâmicos que não podem ser modelados desta forma; nomeadamente os sistemas dinâmicos *contínuos* associados a modelos físicos ou biológicos; estes são descritos, normalmente, por equações diferenciais.

Mesmo no contexto estrito dos sistemas discretos existem duas formas de abordagem à sua descrição: a abordagem *operacional* (aka *simulação)* e a abordagem *lógica.*

Na versão "operacional" o sistema é descrito por um algoritmo cujo comportamento **simula** o do sistema. Um tal abordagem usa, quase sempre, uma única variável do programa para descrever cada variável do modelo, mas como funciona por simulação, só lida com comportamento finitos.

A abordagem "lógica" , usada neste curso, consegue descrever tanto comportamento finitos como infinitos através da verificação da validade de certos juízos lógicos (por exemplo, se uma determinada fórmula é ou não satisfazível). Esta é a bordagem usada neste capítulo como, em grande medida, foi a abordagem usada no capítulo anterior.

Neste capítulo vamos restringir a análise dos sistemas dinâmicos discretos a uma única técnica lógica: o chamado "SAT Model-Checking" em que toda a verificação se baseia, essencialmente, em algoritmos de SAT: toda a formulação lógica do comportamento do sistema tem de ser feita de forma a que as propriedades desse comportamento sejam verificáveis com SAT.

Os modelos lógicos de sistemas dinâmicos têm usualmente um conjunto básico de variáveis X mas a descrição das relações de transição e a formulação das propriedades do comportamento exige um elevado número de cópias ("clones") das variáveis base. Além disso cada transição tem de prever alterações de valor em qualquer estado do sistema; isso faz com que o número de variáveis base seja, à partida, grande. Num sistema complexo o número de variáveis base e o número de "clones" de cada uma dessas variáveis, pode conduzir a um tamanho de fórmulas que tornam quase inviável a verificação SAT.

Por isso, temos de considerar formas alternativas de descrever a lógica dos sistemas dinâmicos que não requeiram "clones" das variáveis base nem requeiram uma representação global do sistema via uma única gigantesca relação de transição. Esta abordagem baseia-se na substituição de relações de transição globais por modelos locais (***"pré-condições mais fracas"*** *e* ***"pós-condições mais fortes"****)* que apresentaremos num dos próximos capítulos.

"SAT Model-Checking" (SAT-MC)

Na continuação da análise das propriedades dinâmicas de FOTS vamos estender algumas a análise feita no capítulo anterior derivada de comportamentos em traços infinitos.

A verificação de propriedades lógicas de sistemas dinâmicos feita em comportamentos ilimitados cinge-se, quase sempre, a propriedades de segurança da forma $\mathbf{G}\,\mathrm{P}$ em que P é um predicado nas variáveis de estado do sistema.

As primeiras técnicas que vimos no Capítulo 3 e que se podem enquadrar nesta definição são as técnicas de *indução* e suas derivadas: *k-indução*. Recordemos que a formalização destas propriedades exigia o uso de operadores de quantificação sobre as variáveis do sistema.

Por exemplo, num FOTS $\Sigma \equiv \langle \mathbf{X}, \mathbf{init}, \mathbf{trans} \rangle$ a verificação de uma propriedade de segurança $\mathbf{G}\,\mathrm{P}$ usando indução simples é feita verificando a validade de duas fórmulas quantificadas:

- $\forall\, s \cdot \mathbf{init}(s) \rightarrow \mathrm{P}(s)$

- $\forall\, s, s' \cdot \mathrm{P}(s) \wedge \mathbf{trans}(s, s') \rightarrow \mathrm{P}(s')$

Verificação usando algoritmos de SAT

Os "provers" SMT não conseguem verificar a validade de fórmulas quantificadas genéricas. Porém conseguem decidir se uma fórmula P não quantificada e com variáveis livres x é ou não **satisfazível (sat) .**

Adicionalmente, se P for satisfazível, então o algoritmo de **SAT** também calcula *testemunhas* ou *provas* de satisfação; isto é calcula um **modelo** (ou **atribuição** de valores a variáveis) $m \equiv \{x \leftarrow v\}$ tais que se verifica $m \vDash P$.

Note-se que uma fórmula satisfazível pode ter vários modelos; **SAT** é um algoritmo não determinístico que produz um desses modelos. Representamos por **SAT+** o algoritmo determinístico que produz **todos** os modelos.

Quando o algoritmo de **SAT** é invocado com a fórmula P como argumento

$$m \leftarrow \mathbf{SAT}(P)$$

representamos $m = \emptyset$ ou por $m = \perp$ o caso em que a fórmula P é insatisfazível; se P for satisfazível então m é não deterministicamente um dos modelos de P .

Genericamente o algoritmo **SAT** permite responder a várias questões:

1. *A fórmula P é uma **taulologia*** i.e. é válido
   $$\vDash \forall\, x \cdot P$$
   se e só se $\neg P$ é **unsat**.
   Nomeadamente, se $m \leftarrow \mathbf{SAT}(\neg P)$ e $m \neq \emptyset$, então obtem-se uma prova de que P não é uma tautologia.
   Provas de satisfação da negação de P designam-se por **contra-exemplos** de P .

2. *A fórmula $P \rightarrow Q$ é uma **tautologia***, i.e. é válido
   $$\vDash \forall\, x \cdot P \rightarrow Q$$
   se e só se $P \wedge \neg Q$ é **unsat.**
   Nomeadamente, se $m \leftarrow \mathbf{SAT}(P \wedge \neg Q)$ e $m \neq \emptyset$, então obtem-se uma prova de que $P \rightarrow Q$ não é tautologia.

Notação “livre de variáveis”

Considere-se de novo contexto de um FOTS $\Sigma \equiv \langle \mathrm{X}, \mathbf{init}, \mathbf{trans} \rangle$ e procuremos escrever com invocaçoes do algoritmo **SAT** a formalização do princípio de indução simples que vimos atrás
para verificar a validade da fórmula temporal $\mathbf{G}\,\mathrm{P}$.

- $\models \forall\, s \cdot \mathbf{init}(s) \rightarrow \mathrm{P}(s)$
- $\models \forall\, s, s' \cdot \mathrm{P}(s) \wedge \mathbf{trans}(s, s') \rightarrow \mathrm{P}(s')$

Esta formalização é fortemente dependente das variáveis que são usadas na quantificação. Recorde-se que a invocação do **SAT** é feita para fórmulas onde ocorre um conjunto *fixo* de variáveis *livres.* Ao invés as fórmulas
$\mathbf{init}(s)$, $\mathrm{P}(s)$, $\mathrm{P}(s')$, $\mathbf{trans}(s, s')$ são determindas por variáveis *ligadas* aos operadores de quantificação.

Por isso é necessário adaptar o formalismo inicial da indução ao modo como as variáveis são tratadas na metodologia **SAT**.

Vamos considerar um operador $\mathbf{next}$ que cria variáveis novas como “clones” de variáveis existentes. Este operador está implícito na definição inicial de um FOTS quando se considera o conjunto de variáveis X' como “clones” das variáveis iniciais do sistema X; de facto tem-se

$$\mathrm{X}' \equiv \mathbf{next}(\mathrm{X})$$

Note-se que

> *O operador **next** é determinístico: aplicado duas vezes ao mesmo argumento produz duas vezes o mesmo resultado.*

No contexto da “package” **pysmt** a criação de variáveis e a funcionalidade do operador $\mathbf{next}$ são implementados pelo par de funções `pysmt.shortcuts.Symbol` e `pysmt.shortcuts.FreshSymbol`.
Com a primeira destas funções são criadas as variáveis base. Com a segunda função são criados os “clones”.

Existe um diferença fundamental entre **next** e `pysmt.shortcuts.FreshSymbol`. O operador **next** é determinístico, mas o operador `pysmt.shortcuts.FreshSymbol`

não é: cada invocação desta função produz uma variável diferente mesmo que seja aplicado aos mesmos argumentos.

Na notação que vamos usar neste capítulo vamos designar o predicado que representa os estados iniciais por $\mathbf{I}$.
A fórmula $\mathbf{I}$ é um predicado nas variáveis livres X. Se quisermos representar a mesma fórmula substituindo as variáveis X por X', então usamos a notação $\mathbf{I}'$.
Isto é

$$\mathbf{I}' \equiv \mathbf{I}\{\mathrm{X}/\mathbf{next}(\mathrm{X})\}$$

Para todo predicado P na variável X, usa-se a notação $\mathrm{P}\{\mathrm{X}/\mathrm{Z}\}$ para representar o predicado que se obtém substituindo em P a variável X pela expressão Z.

A mesma notação aplica-se a um qualquer predicado P que tenha X como o seu conjunto de variáveis livres: representa-se por P' a fórmula nas variáveis livres X' que se obtém de P substituindo que cada variável X pela variável correspondente X'.

$$\mathrm{P}' \equiv \mathrm{P}\{\mathrm{X}/\mathbf{next}(\mathrm{X})\}$$

Exemplo 1
Vamos usar $\mathbf{T}$ o predicado que tem variáveis livres X e X' e que representa a relação de transição. Com estas convenções a formalização por indução simples da validade de $\mathbf{G}\,\mathrm{P}$ será

- $\mathrm{SAT}(\mathrm{I} \wedge \neg \mathrm{P}) = \emptyset$
- $\mathrm{SAT}(\mathrm{P} \wedge \mathrm{T} \wedge \neg \mathrm{P}') = \emptyset$

A notação segue as seguintes convenções:

- Todos os predicados unários são expressões que têm livres as variáveis X.
- Todos os predicados binários são expressões que têm livres as variáveis X e X'.
- Todo o predicado *n*-ário é uma expressão que tem livres variáveis $\mathrm{X}^{(0)}, \mathrm{X}^{(1)} \cdots \mathrm{X}^{(n-1)}$ com
$$\mathrm{X}^{(0)} \equiv \mathrm{X} \quad \text{e} \quad \mathrm{X}^{(i+1)} \equiv \mathbf{next}(\mathrm{X}^{(i)}) \quad \text{para } i > 0$$
- Se P é um predicado *n*-ário então
$$\mathrm{P}' \equiv \mathrm{P}\{\mathrm{X}^{(i)}/\mathrm{X}^{(i+1)} \text{ para } i < n-1,\ \mathrm{X}^{(n-1)}/\mathbf{next}(\mathrm{X}^{(n-1)})\}$$

Isto é: cada uma das primeiras $n-1$ variáveis é substituída pela variável seguinte na lista; a última variável é substituída por um novo "clone" dessa mesma variável.

Por conveniência, em todas estas sequências de variáveis $\mathrm{X}^{(0)}$ coincide com a variável X que ocorre na definição de Σ e as restantes variáveis resultam de sucessivas invocação de $\mathbf{next}(\mathrm{X})$,

Nomeadamente a variável $\mathrm{X}^{(i)}$ resulta da *i*-ésima invocação de **next**(X). Pode-se interpretar a sequência de variáveis $\{\mathrm{X}^{(i)}\}_{i\geq 0}$ como um dicionário que é construída um só vez ; todas as invocações de um $\mathrm{X}^{(i)}$ específico obtém-se consultando esse dicionário com a chave i.

Adicionalmente vamos considerar um função **top** que aplicado a um predicado n -ário P dá como resultado a <u>última</u> variável na sequência de variavéis que ocorrem em P.

Exemplo

Uma sequência de transições

$$s_0 \xrightarrow{T} s_1 \xrightarrow{T} s_2 \cdots s_{n-1} \xrightarrow{T} s_n$$

pode ser definida indutivamente da como potências da relação binária T

$$\mathrm{T}^{n+1} \equiv \mathrm{T} \wedge (\mathrm{T}^n)' \quad \text{para} \quad n > 0$$

Tem-se também

$$\mathbf{top}(\mathrm{T}^n) = \mathbf{next}^n(\mathrm{X})$$

Se esta definição for aplicada a um predicado unário P vemos que se tem uma definição de uma sequência *n*-ária P^{n+1} pelas relações

$$\mathrm{P}^0 \equiv \mathrm{True} \qquad \text{e} \qquad \mathrm{P}^{n+1} \equiv \mathrm{P} \wedge (\mathrm{P}^n)' \qquad \text{para } n \geq 0$$

Finalmente note-se que $\mathrm{P} \wedge \mathrm{T}$ é também um predicado binário. A esse predicado pode-se aplicar (tal como a T) a *n*-ésima potência .

Pelas definições atrás e pelo facto do operador **next ser determinístico** é simples verificar que

$$(\mathrm{P} \wedge \mathrm{T})^n \equiv \mathrm{P}^n \wedge \mathrm{T}^n$$

Vimos que uma notação precisa que abstrai em relação à complexidade das variáveis simplifica a escrita das prova por indução simples. Para a descrição de uma qualquer técnica, a ecrita precisa da sua especificação é meio caminho andado para uma implementação computacional simples e livre de erros.

Por exemplo, a definição da verificação por k-**indução** da propriedade $\mathrm{G}\,\mathrm{P}$ pode ser escrita nas ho condições:

- $\mathrm{SAT}(\mathrm{I} \wedge \mathrm{T}^{j-1} \wedge \neg \mathrm{P}^{j}) = \varnothing$ para todo $j < k$
- $\mathrm{SAT}(\mathrm{P}^{k} \wedge \mathrm{T}^{k} \wedge \neg \mathrm{P}^{k+1}) = \varnothing$

Segurança e Acessibilidade

A verificação de propriedades lógicas de sistemas dinâmicos feita em comportamentos ilimitados cinge-se, quase sempre, a propriedades de segurança da forma $\mathrm{G}\,\mathrm{P}$ em que P é uma **propriedade de segurança** expressa como um predicado nas variáveis de estado do sistema.
Como não é possível verificar fórmulas temporais genéricas, muitos modelos de sistemas dinâmicos incluem a propriedade P diretamente na formulação do próprio sistema e dispensam a lógica temporal.
Mais exatamente é mais natural exprimir a propriedade de segurança P como a negação de uma **condição de erro** E; isto é, o sistema está num estado seguro se não estiver num estado onde se verifique a condição de erro E.

Assim os designados **safe first order transitions systems (SFOTS)** são tuplos

$$\Sigma \equiv \langle \mathrm{T}, \mathrm{X}, \mathrm{next}, \mathrm{I}, \mathrm{T}, \mathrm{E} \rangle$$

Recorde-se que na definição de um FOTS é essencial fixar qual é o fragmento da FOL onde o modelo se desenvolve; esse fragmento lógico é descrito numa SMT apropriada T. Em seguida a definição específica as variáveis base X, o operador next que gera os vários "clones" dessas variáveis, o predicado unário I que determina os estados iniciais e o predicado binário T que define a relação de transição. Um SFOTS contém todos esses items e ainda um predicado unário E que define uma **condição de erros**.

O algoritmo de verificação num SFOTS tem objetivos distintos da verificação usada nos FOTS.
Vimos no Capítulo 3 que o algoritmo de verificação da validade de propriedades dinâmicas de um FOTS usa dois argumentos: o *comportamento dinâmico* de um sistema Σ (i.e. o conjunto dos seus traços) e a *propriedade temporal* ϕ cuja validade se quer verificar nesse comportamento.
O algoritmo decide sobre a validade de um *juízo* da forma $\Sigma \models \phi$.

Ao invés num SFOTS a propriedade de segurança (ou condição de erro) está contida na própria definição do sistema. A especificação do sistema determina o *comportamento* Σ formado pelos traços mas também o subconjunto $\Sigma_U \subseteq \Sigma$ formado pelos traços que violam a condição de segurança. Agora o algoritmo de verificação limita-se a decidir se o conjunto Σ_U é ou não vazio.

Para mais detalhes são necessárias algumas noções adicionais

Num SFOTS $\Sigma \equiv \langle \mathrm{X}, \mathrm{next}, \mathrm{I}, \mathrm{T}, \mathrm{E} \rangle$ a **verificação** derivado das noções de **acessibilidade** e **insegurança.**

> *Um estado r é **acessível ("reachable")** em Σ quando $r \in \mathrm{I}$ ou quando existe uma transição $(s, r) \in \mathrm{T}$ em que s é acessível em Σ.*

> *Um estado u é **inseguro ("unsafe")** em Σ quando $u \in \mathrm{E}$ ou quando existe uma transição $(u, v) \in \mathrm{T}$ em que v é inseguro em Σ.*

> *O SFOTS Σ é **inseguro** se existe algum estado s que seja simultaneamente acessível e inseguro. Em caso contrário o sistema Σ é seguro.*

Por motivos que serão óbvios em seguida, é conveniente definir um novo SFOTS

$$\Sigma^{\top} \equiv \langle \mathrm{Y}, \mathrm{next}, \mathrm{E}, \mathrm{B}, \mathrm{I} \rangle$$

em que

a. A variável Y é um "clone" da variável X; invocações sucessivas de $\mathbf{next}(\mathrm{Y})$ produzem variáveis
b. troca-se os papéis da condição inicial e da condição de erro; isto é, E é a nova condição inicial, enquanto que I é a nova condição de erro,

c. a relação de transição B é a inversa T^{-1} da relação inversa de T; isto é,

$$(r, s) \in \mathrm{B} \quad \text{se e só se} \quad (s, r) \in \mathrm{T}$$

Representado T por um predicado binário nas variáveis X e X', então o predicado que representa a inversa B obtem-se trocando em T as variáveis: X é substituído por X' e X' é substituído por X.

Exemplo
Uma atribuição $x \leftarrow x - 1$, numa linguagem imperativa, é representada pela fórmula

$$\mathrm{T} \equiv (x = x' + 1)$$

A transição inversa será

$$\mathrm{B} \equiv \mathrm{T}^{-1} \equiv (x' = x + 1)$$

Atendendo às definições de acessibilidade e insegurança conclui-se facilmente

> ***Facto***
> *Um estado s é inseguro em Σ se e só se s é acessível em $\Sigma^{\top}$. Um estado s é acessível em Σ se e só se s é inseguro em $\Sigma^{\top}$.*

Este resultado permite-nos analisar as condições de segurança exclusivamente através das relações de acessibilidade em Σ e $\Sigma^{\top}$.

Como caracterizar os estados acessíveis e proibidos?

A definição dos estados acessíveis R e dos estados proibidos U permite construir uma definição formal destes conjuntos

a. R é <u>o menor conjunto/predicado</u> que verifica
   i. $\mathrm{I} \wedge \neg R$ é **unsat**
   ii. $R \wedge \mathrm{T} \wedge \neg R'$ é **unsat**
b. U é <u>o menor conjunto/predicado</u> que verifica
   i. $\mathrm{E} \wedge \neg U$ é **unsat**
   ii. $U \wedge \mathrm{T}^{-1} \wedge \neg U'$ é **unsat**

Estas condições são precisamente as mesmas que são usadas, nas provas por indução, para verificar se R é invariante em Σ e U é invariante no dual $\Sigma^\top$.

É possível computar exatamente os conjuntos R ou U a partir destas definições? Equivalentemente, é possível computar o menor invariante de Σ ou $\Sigma^\top$?

Infelizmente, para a maioria dos casos, a resposta é *não*. A razão está no qualificativo "o menor que..." . Isto porque, sendo R e U conjunto infinitos ou muito grandes não é computável minimizar R ou U a partir das condições definidoras.

A alternativa é computar R e U de forma aproximada. Isto é calcular limites superiores $\overline{R} \supseteq R$ e $\overline{U} \supseteq U$ que também sejam invariantes ; desta forma

- se $\overline{R} \wedge \overline{U}$ for **unsat** também $R \wedge U$ será **unsat** e concluimos que o sistema é seguro
- se $\overline{R} \wedge \overline{U}$ for **sat** e se for s um contra-exemplo não se pode concluir se $R \wedge U$ é ou não é **unsat;** é preciso construir melhores aproximações; o conhecimento de s pode contribuir para uma melhor resposta ser obtida.

Em alternativa pode-se considerar aproximações inferiores: $\mathrm{R}_n \subseteq \mathrm{R}$ e $\mathrm{U}_m \subseteq \mathrm{U}$.

Neste caso se, para para algum par R_n , U_m se verificar uma interseção de conjuntos, o sistema é grantidamente inseguro. Se tal não acontecer é preciso construir melhores aproximações.

1. Pode-se começar por caracterizar um traço finito com n transições em Σ escrito como

   $$\mathrm{R}_n \equiv \mathrm{I} \wedge \mathrm{T}^n \qquad \text{e} \qquad \mathrm{R} \equiv \bigvee_{n=0}^{\infty} \mathrm{R}_n$$

   Usando um algoritmo **SAT** qualquer modelo modelo $M_n \leftarrow \mathbf{SAT}(\mathrm{R}_n)$ produz uma atribuição de valores às variáveis $\mathrm{X}_0, \cdots, \mathrm{X}_n$ que descrevem estados acessíveis com *n* ou menos transições.

2. Dualmente, usando o sistema $\Sigma^\top$ um traço finito com m transições é escrito como

   $$\mathrm{U}_m \equiv \mathrm{E} \wedge (\mathrm{T}^{-1})^m \qquad \text{e} \qquad \mathrm{U} \equiv \bigvee_{m=0}^{\infty} \mathrm{U}_m$$

Usando **SAT** qualquer modelo $M_m^{\top} \leftarrow \mathbf{SAT}(\mathrm{U}_m)$ determina uma atribuição de valores de estado às variáveis $\mathrm{Y}_0, \cdots, \mathrm{Y}_m$. Tais estados são sempre inseguros porque um estado de erro é acessível a partir desses estados em m ou menos transições T.

3. Para avaliar a segurança eventual do sistema Σ é necessário determinar se nenhum estado é simultâneamente acessível e inseguro. Para isso temos de avaliar se, para <u>todo</u> o número de passos de acessibilidade n e para <u>todo</u> número de passos de acessibilidade de insegurança m pode-se definir o predicado "query" $\mathrm{Q}_{n,m}$ como

$$\mathrm{Q}_{n,m} \equiv \mathrm{R}_n \wedge (X_n = Y_m) \wedge \mathrm{U}_m$$

sendo $X_n = \mathbf{top}(\mathrm{R}_n) = \mathbf{next}^n(\mathrm{X})$ e $Y_m = \mathbf{top}(\mathrm{U}_m) = \mathbf{next}^m(\mathrm{Y})$, e verificar

$$\forall\ n, m \cdot \mathbf{SAT}(\mathrm{Q}_{n,m}) = \varnothing$$

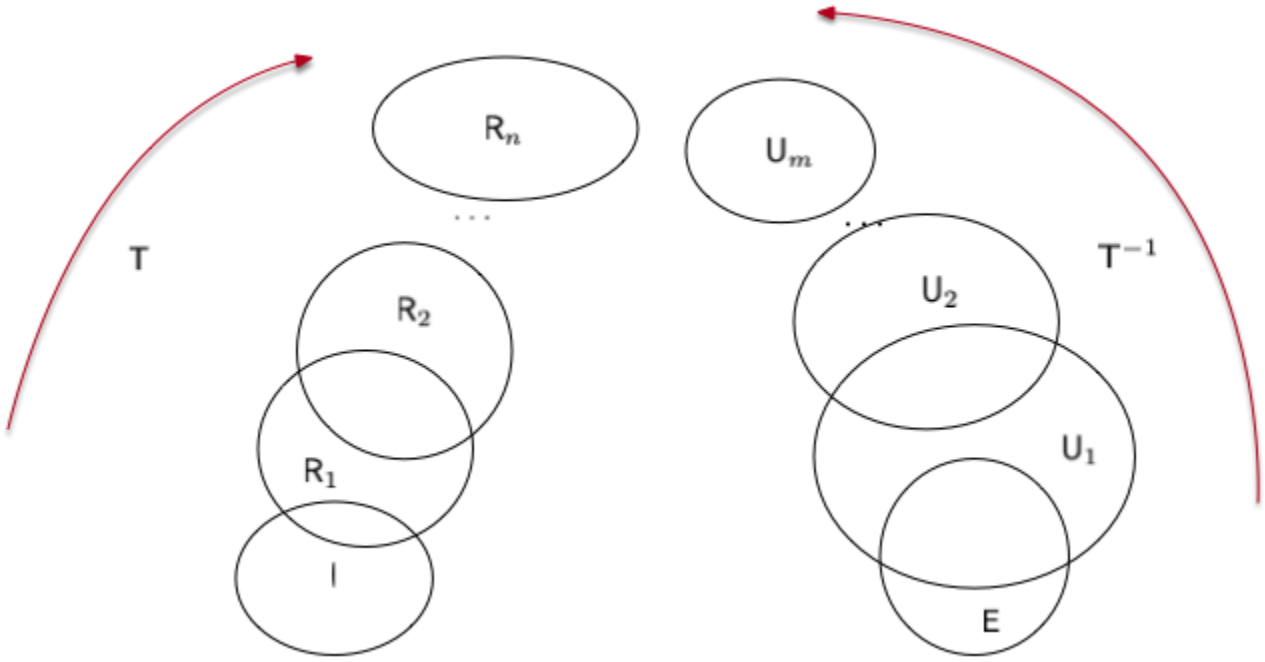
3. Note-se que, como R_n e U_m usam variáveis distintas é necessário introduzir o termo $(X_n = Y_m)$ para forçar que o topo de ambas as sequências instanciem nos mesmo estados.

4. A condição acima tem de se verificar <u>para todos</u> n, m . Como não é possível construir um algoritmo que percorra um número infinito de pares (n, m) , testando para cada par a condição acima, é necessário procurar uma solução <u>finita</u> que verifique um número infinito de testes. É preciso construir uma **aproximação finita** a esta condição.

Recorde-se que o princípio da indução faz precisamente este tipo de função: obter condições finitas que sejam suficientes para garantir a validade de um número infinito de testes.

Antes vimos outra aproximação: o **BMC**. Nesta abordagem fixávamos um valor máximo para os índices m e n , ignorando os testes a "queries" $\mathrm{Q}_{n,m}$ com índices superiores a esse valor.

Todo o **algoritmo de verificação de segurança** deve essencialmente este princípio; nas próximas seções vamos apresentar algumas estratégias para construção de um tal algoritmo.



Safe First Order Transition System: Reachability and Unsafety

---

Interpolantes e Invariantes

Um algoritmo para verificação de propriedades de segurança de um SFOTS em traços ilimitados tem de definir um número finito de de testes SAT que permitam verificar um número infinito de condições lógicas: normalmente uma condição por cada estado do traço.

As técnicas de k-indução são uma abordagem a esta questão mas muitas vezes não é possível usar de forma eficiente esta abordagem.

Uma segunda abordagem requer algumas noções específicas, as noções de **interpolante** e a de **invariante,** que vamos descrever em seguida.

Interpolantes

Sejam $A(x,w)$ e $B(w,y)$ fórmulas de 1ª ordem em que x é uma sequência de variáveis que ocorrem só em A, y é uma sequência de variáveis que ocorrem só em B e w é a sequência de variáveis que ocorrem em A e em B.

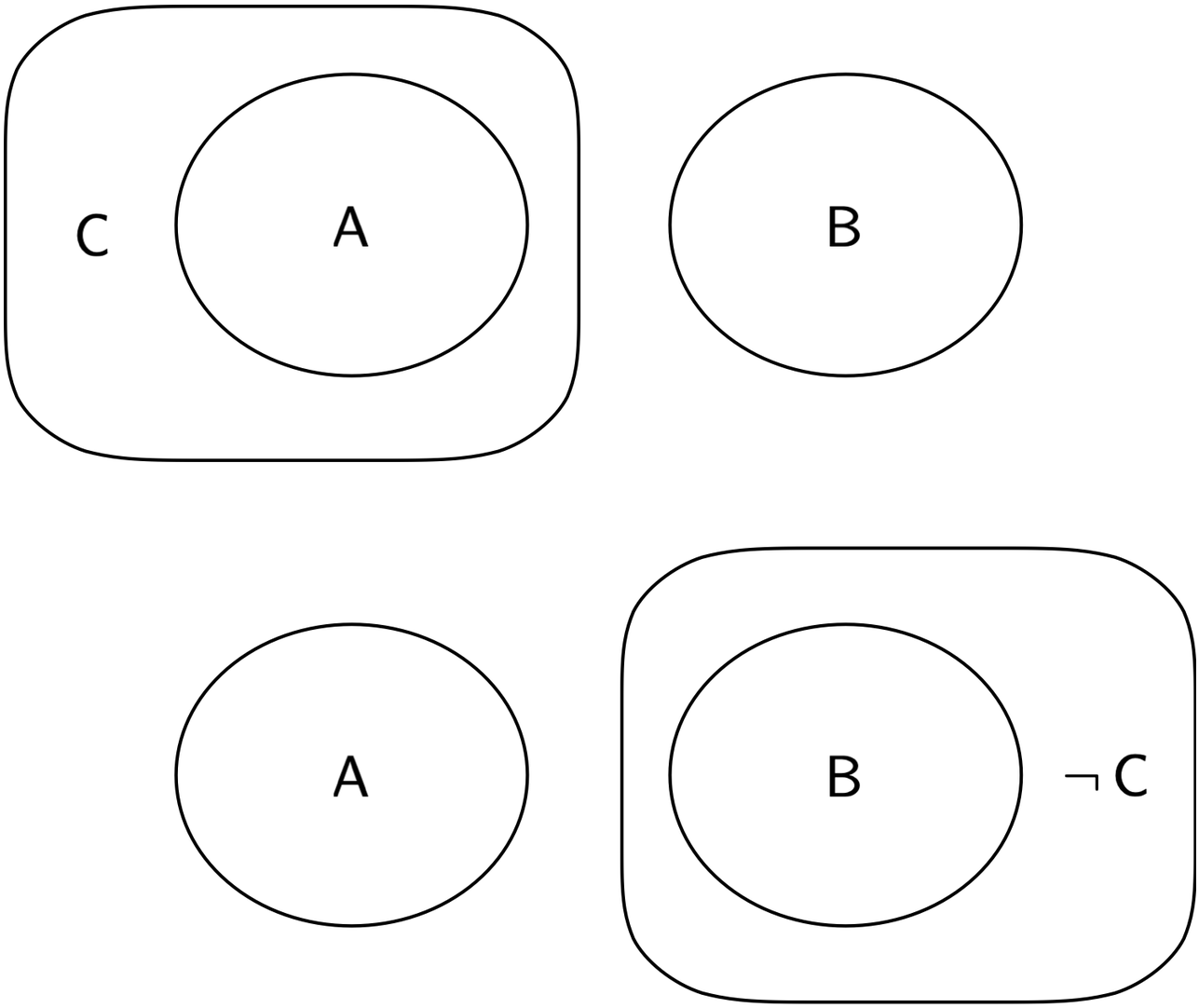
Um **interpolante** para o par (A,B) é definível quando A e B são mutuamente inconsistentes (itsto é, $A \wedge B$ é insatisfazível) e, nesse caso, é uma fórmula de 1ª ordem C tal que:

- C só contém as variáveis comuns a A e a B; neste caso serão as variáveis w.
- Os pares $(A, \neg C)$ e (C,B) são ambos mutuamente inconsistentes.

Existe um teorema, o **teorema de Craig**, que permite construir de forma eficiente eficiente esse interpolante. A demonstração deste teorema é feita por indução e fornece o algoritmo básico para calcular interpolantes. Não vamos aqui apresentar o algoritmo em detalhe mas convém referir que os "solvers" SMT mais comuns (e.g. Z3, MathSAT, ...) fornecem implementações dos interpolantes de Craig.

Da definição conclui-se que

> *C é um interpolante para o par (A,B) se e só se $\neg C$ é interpolante para o par (B,A).*



Iremos usar os interpolantes como "aproximações superiores" dos predicados R_n e U_m que vimos anteriormente. Considere-se, por exemplo, o predicado que representa

$$\mathrm{Q}_{n,m} \equiv \mathrm{R}_n \wedge (X_n = Y_m) \wedge \mathrm{U}_m$$

O termo R_n contém a sequência de variáveis livres $X_0, \cdots, X_n$, aqui representada por $\overline{X}_n$. O termo U_m contém a sequência de variáveis livres $Y_0, \cdots, Y_n$, aqui representada por $\overline{Y}_m$. Estes dois conjuntos de variáveis são disjuntos; só o terceiro termo, o predicado $(X_n = Y_n)$, contém variáveis comuns a R_n e a U_m.

A verificação da segurança do SFOTS consiste em que provar que, <u>para todos</u> n, m, a fórmula $Q_{n,m}$ (que tem livres as variáveis $\overline{X}_n$ e as variáveis $\overline{Y}_m$) é insatisfazível.

$$\forall\, n, m \cdot \mathrm{SAT}(Q_{n,m}) = \varnothing$$

Para provar que $\mathrm{Q}_{n,m}$ é insatisfazíevel para todo n, m vai-se usar uma estratégia similar ás provas por indução. Para tal precisamos de recorrer a interpolantes e a invariantes.

Nomeadamente os interpolantes intervêm considerando que um particular $V_{n,m}$ é insatisfazível. Então existe um interpolante calculável pelo algoritmo de Craig. Nomeadamente, fazendo

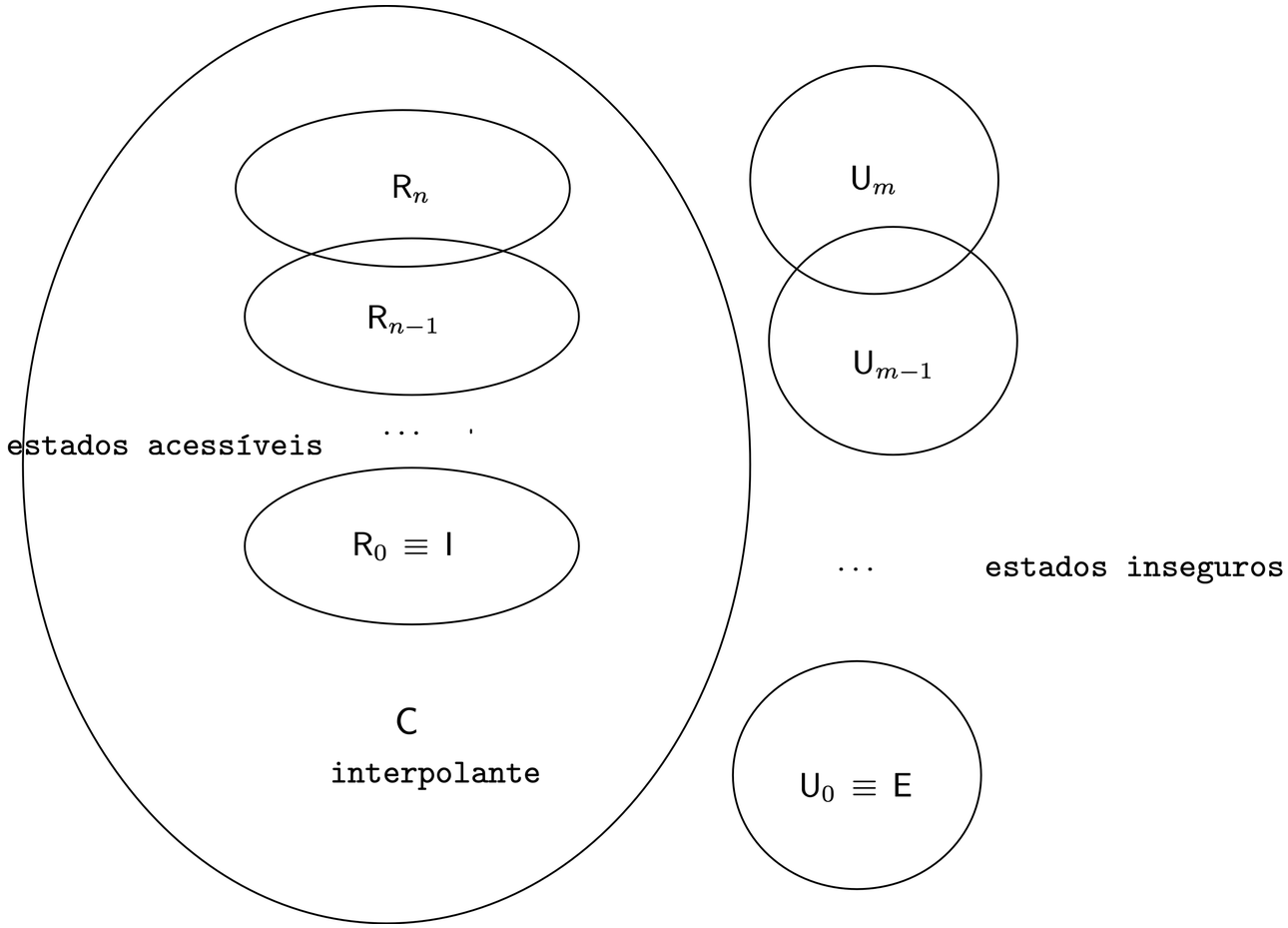
$$A \equiv R_n \wedge (X_n = Y_m) \quad \text{e} \quad B \equiv U_m$$

e notando que Y_m é a única variável comum a A e a B, o algoritmo constrói $C(Y_m)$ tal que

a. C só contém livre a variável Y_m;
   Isto porque Y_m é a única variável que aparece simultaneamente em A e B.

b. Pela definição de interpelante verificam-se duas condições importantes
   - $A \wedge \neg C(Y_m)$ é **unsat**
     Como $A \equiv R_n \wedge (X_n = Y_m)$ conclui-se que $R_n \rightarrow C(X_n)$ é tautologia e portanto, como conjuntos, $R_n \subseteq C(X_n)$

   - $B \wedge C(Y_m)$ é **unsat**
     como $B \equiv U_m$ conclui-se que, como conjuntos, $U_m \cap C(Y_m) = \varnothing$



Invariantes

Na sua formulação mais simples, os invariantes são predicados cujo valor lógico se mantém após a execução da transição T. Na formalização do princípio da **indução simples**, um predicado P é **invariante** quando

$$\mathrm{P} \wedge \mathrm{T} \rightarrow \mathrm{P}' \qquad \text{é tautologia}$$

ou, equivalentemente,

$$\mathbf{SAT}(\mathrm{P} \wedge \mathrm{T} \wedge \neg \mathrm{P}') = \varnothing$$

A generalização da indução simples é a k**-indução.** Neste contexto P será k-**invariante** se

$$(\mathrm{P} \wedge \mathrm{T})^k \rightarrow \mathrm{P}^{k+1} \qquad \text{é tautologia}$$

ou, equivalentemente,

$$\mathbf{SAT}(\mathrm{P}^k \wedge \mathrm{T}^k \wedge \neg \mathrm{P}^{k+1}) = \varnothing$$

Estas construções podem ser generalizadas para qualquer predicado n-ário F nas variáveis $X_0, \cdots, X_n$, com $n \geq 1$ e, como é usual, X_0 coincide com a variável geradora X .

Neste contexto o predicado P é **invariante** quando, sendo $X_n \equiv \mathbf{top}(\mathrm{F})$, se verifica

$$\mathrm{P} \ \wedge \ \mathrm{F} \ \rightarrow \ \mathrm{P}\{\mathrm{X}/X_n\} \quad \text{é tautologia}$$

ou, equivalentemente,

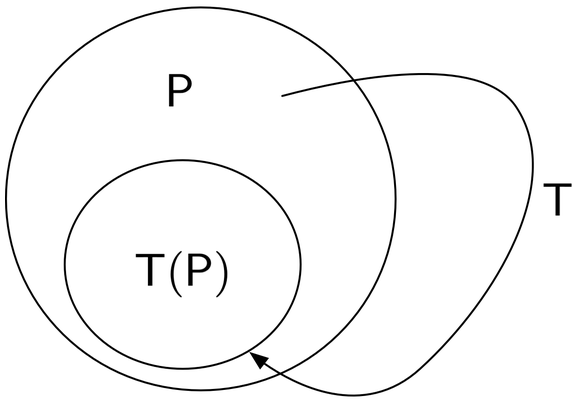
$$\mathbf{SAT}(\mathrm{P} \ \wedge \ \mathrm{F} \ \wedge \ \neg \mathrm{P}\{\mathrm{X}/X_n\}) \ = \ \varnothing$$

Finalmente se P, Q são ambos predicado em X, diz-se que P é **invariante relativo** a Q para a transição T quando é tautologia

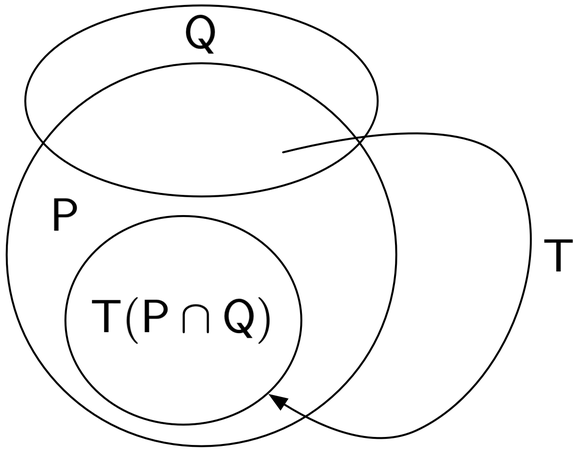
$$\mathrm{Q} \ \wedge \ \mathrm{P} \ \wedge \ \mathrm{T} \ \rightarrow \ \mathrm{P}'$$

A verificação, usando SAT, desta tautologia e a sua extensão para outras noções de invariância realiza-se como nos casos anteriormente descritos.

A noção de **invariância relativa** estende a noção mais simples adicionando um predicado extra no antecedente da transição. Isto é, para garantir que o consequente da transição verifica a propriedade P, é preciso garantir que o antecedente também verifica P e, além disso, verifica a propriedade extra Q.



invariante



invariante relativo

Que relação existe entre os invariantes de T e os invariantes da relação inversa $\mathrm{B} \equiv \mathrm{T}^{-1}$.

Note-se que, para um predicado unário P ser um invariante de T, tem-se

$$(\forall s, r \cdot \mathrm{P}(s) \wedge \mathrm{T}(s, r) \wedge \neg \mathrm{P}(r)) \quad \equiv \quad \bot$$

Como $\mathrm{T}(s, r) \equiv \mathrm{B}(r, s)$ então a relação anterior pode ser escrita como

$$(\forall s, r \cdot \neg \mathrm{P}(r) \wedge \mathrm{B}(r, s) \wedge \neg(\neg \mathrm{P}(s))) \quad \equiv \quad \bot$$

Isto significa que

> *O predicado* P *é um invariante de* T *se e só se o predicado* $\neg \mathrm{P}$ *é um invariante de* B.

"Model-Checking" usando interpolantes e invariantes

Considere-se de um SFOTS $\Sigma \equiv \langle X, \mathrm{I}, \mathrm{T}, \mathrm{E} \rangle$. Vamos supor que o sistema não é trivialmente inseguro; nomeadamente supomos que $\mathrm{SAT}(\mathrm{I} \wedge \mathrm{E}) = \varnothing$.

Considere-ase de novo as fórmulas R_n , U_m e

$$\mathrm{Q}_{n,m} \equiv \mathrm{R}_n \wedge (X_n = Y_m) \wedge \mathrm{U}_m$$

como vimos anteriormente. O algoritmo de "model checking" pretende verificar se para todo n, m a fórmula $\mathrm{Q}_{n,m}$ é insatisfazível.

O algoritmo derivado das noções de interpolante e invariante baseia-se no seguinte resultado

> *Se existe um predicado unário* C *que é invariante de* T *e, para algum par de índices* (n, m) *verifica-se que são tautologias* $\mathrm{R}_n(\overline{X}_n) \rightarrow C(X_n)$ *e* $\mathrm{U}_m(\overline{Y}_m) \rightarrow \neg C(Y_m)$ *, então* $\mathrm{Q}_{n',m'}$ *é insatisfazível para todo* $n' \geq n$ *e* $m' \geq m$ *.*

Prova
Se C é invariante de T, então $\neg C$ é invariante de $\mathrm{B} \equiv \mathrm{T}^{-1}$.
Assim, por hipótese, ambas as fórmulas $C \wedge \mathrm{T} \rightarrow C'$ e $\neg C \wedge \mathrm{B} \rightarrow \neg C'$ são tautologias.

Suponhamos que n é tal que $\mathrm{R}_n \rightarrow C(X_n)$ é tautologia.
Como $C(X_n) \wedge \mathrm{T}(X_n, X_{n+1}) \rightarrow C(X_{n+1})$ é tautologia, temos as tautologias

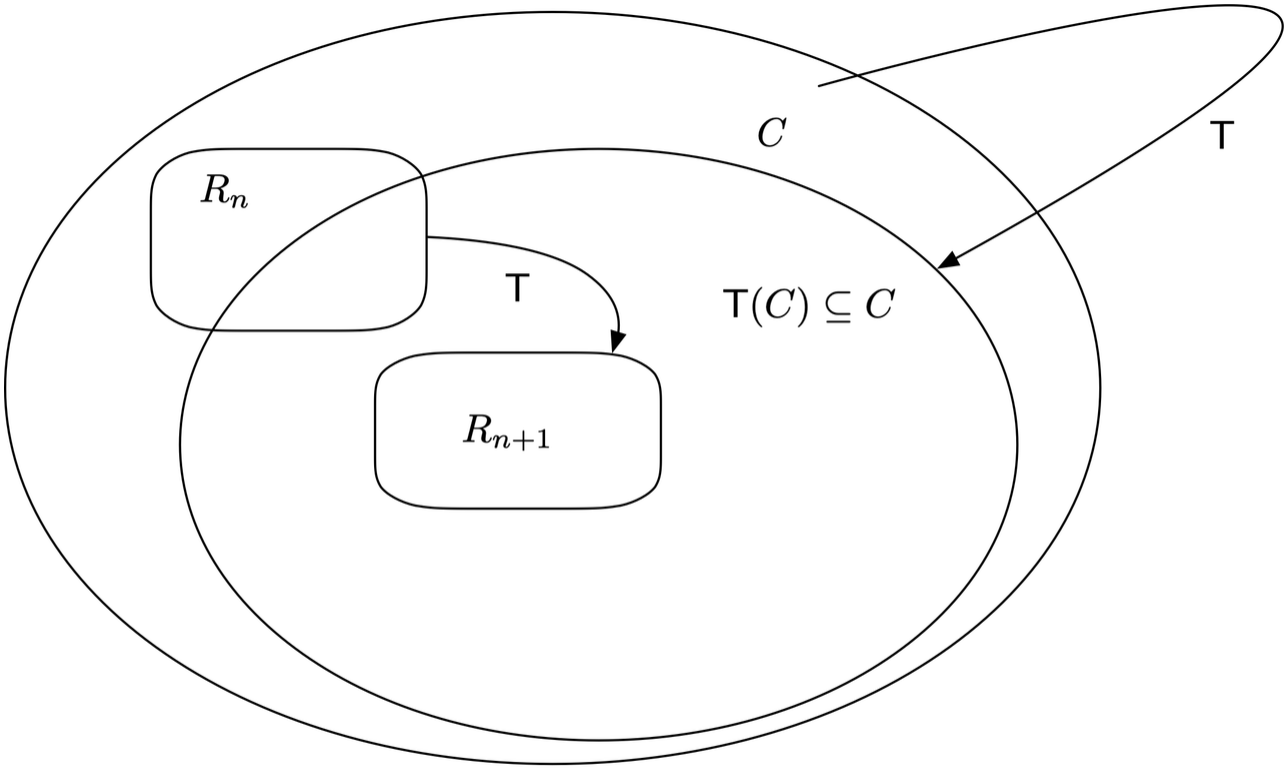
$$R_{n+1} \equiv R_n \wedge \mathrm{T}(X_n, X_{n+1}) \rightarrow C(X_n) \wedge \mathrm{T}(X_n, X_{n+1}) \rightarrow C(X_{n+1})$$

Portanto, por indução simples, verifica-se $\mathrm{R}_{n'} \rightarrow C(X_{n'})$ é tautologia para todo $n' \geq n$.

Para os U_m procede-se do mesmo modo, atendendo que $U_{m+1} \equiv U_m \wedge \mathrm{B}(Y_m, Y_{m+1})$, e prova-se por indução simples que $U_{m'} \rightarrow \neg C(Y_{m'})$ para todo $m' \geq m$.

Portanto, para todos $n' \geq n$ e $m' \geq m$ tem-se

$$\mathrm{Q}_{n',m'} \equiv \mathrm{R}_{n'} \wedge \mathrm{U}_{m'} \wedge (X_{n'} = Y_{m'}) \rightarrow C(X_{n'}) \wedge (X_{n'} = Y_{m'}) \wedge \neg C(Y_{m'}) \rightarrow \perp$$



$$R_n \subseteq C \Rightarrow T(R_n) \subseteq T(C) \Rightarrow R_{n+1} \subseteq C$$

Igualmente $\neg C$ é invariante de T^{-1}; isto é

$$\mathrm{T}^{-1}(\neg C) \subseteq \neg C \quad \text{e} \quad U_{m+1} \equiv T^{-1}(U_m) \subseteq T^{-1}(\neg C) \subseteq \neg C$$

Todos os futuros $R_{n'}$ estão contidos em C e todos os futuros $U_{m'}$ estão contidos em $\neg C$. Portanto nunca se intersectam.

O algoritmo de "model-checking" manipula as fórmulas $\mathrm{R}_n, \mathrm{U}_m$ fazendo crescer os índices n, m de acordo com as seguintes regras

1. Inicia-se $n = m = 0$, $\mathrm{R}_0 \equiv \mathrm{I}$ e $\mathrm{U}_0 \equiv \mathrm{E}$.

2. No estado (n, m) tem-se a certeza que em todos os estados anteriores não foi detectada nenhuma justificação para a insegurança do SFOTS.
   Testa-se se $\mathrm{Q}_{n,m} \equiv \mathrm{R}_n \wedge (X_n = Y_m) \wedge \mathrm{U}_m$ é satisfazível; se tal ocorrer o sistema é inseguro e o algoritmo termina com a mensagem **unsafe** .

3. Se $\mathrm{Q}_{n,m}$ for insatisfazível calcula-se C como o interpolante do par
   $$A \equiv \mathrm{R}_n \wedge (X_n = Y_m) \quad \text{e} \quad B \equiv \mathrm{U}_m .$$
   Neste caso verificam-se as tautologias $\mathrm{R}_n \rightarrow C(X_n)$ e $\mathrm{U}_m \rightarrow \neg C(Y_m)$.

4. Testa-se se C é um invariante de T verificando a condição
   $$C(X_n) \wedge \mathrm{T}(X_n, Y_m) \wedge \neg C(Y_m) \quad \text{é } \mathbf{unsat}$$
   Se C for invariante então, pelo resultado anterior, sabe-se que $\mathrm{Q}_{n',m'}$ é insatisfazível para todo $n' \geq n$ e $m' \geq n$, e por isso o sistema é seguro.
   O algoritmo termina com a mensagem **safe**.

5. Se C não for invariante de T procura-se encontrar um majorante $S \supseteq C$ que verifique as condições do resultado referido: isto é, ser um invariante de T e ser disjunto de U_m .
   Se for possível encontrar tal majorante S — ver rotina em seguida — então o algoritmo termina com a mensagem **safe**.

6. Se não for possível encontrar esse majorante, então pelo menos um dos índices n, m é incrementado, e as fórmulas R_n , U_m são actualizadas da seguinte forma
   i. São criadas novas variáveis X_{n+1} e/ou Y_{m+1} .
   ii. Calcula-se $R_{n+1} \equiv R_n \wedge \mathrm{T}(X_n, X_{n+1})$ e/ou
   $$U_{m+1} \equiv U_m \wedge \mathrm{B}(Y_m, Y_{m+1})$$
   Repete-se o processo a partir do passo 2.

A parte crítica é o passo 5. e é por isso que não está aqui especificado. Várias estratégias são possíveis. Uma solução possível é um algoritmo iterativo que tenta encontrar um invariante S

a. S é inicializado com $C(X_n)$
b. Faz-se $A \equiv S(X_n) \wedge \mathrm{T}(X_n, Y_m)$ e verifica-se se $A \wedge U_m$ é insatisfazível. Se for satisfazível então não é possível encontrar o majorante e esta rotina termina sem sucesso.
c. Se $A \wedge U_m$ for insatisfazível calcula-se um novo interpolante $C^*(Y_m)$ do par (A, U_m).
d. Se $C^*(X_n) \wedge \neg S(X_n)$ for **unsat**, então S é o invariante pretendido e a rotina termina em sucesso retornando S.

   > *Isto porque a definição de interpolante diz-nos que* $S(X_n) \wedge \mathrm{T}(X_n, Y_m) \rightarrow C^*(Y_m)$ *é tautologia; esta condição diz-nos que* $C^*(Y_m) \rightarrow S(Y_m)$ *é também tautologia; logo, por transitividade,* $S(X_n) \wedge T(X_n, Y_m) \rightarrow S(Y_m)$ *será uma tautologia.*

e. Se $C^* \wedge \neg S$ for **sat**, actualiza-se S com

$$S \leftarrow S \vee (C^* \wedge \neg S)$$

e repete-se o processo a partir do passo (b).

Nota
Dado que se pretende que S seja próximo do menor invariante, a actualização de S convém que seja feita de modo a aumentá-lo "pouco". Dado que o algoritmo SAT aplicado a $C^* \wedge \neg S$ produz um ou mais modelos m que verificam $m \vDash (C^* \wedge \neg S)$, basta actualizar $S(X)$ para $S(X) \vee (X = m)$.