Capítulo 3 : “Satisfiability Modulo Theories” (2ª Parte)

Sistemas de Transição de 1ª Ordem (FOTS)

Sistemas de Transição de 1ª Ordem (“First Order Transition Systems” ou FOTS), são modelos de sistemas dinâmicos que, tal como as FSM, são determinadas por um espaço de estados, uma relação de transição e um conjunto de estados iniciais.

Em relação às FSM’s as diferenças essenciais são

- O espaço de estados é, em princípio, infinito e está contido num fragmento decidível da LPO. Na caracterização do espaço de estados está sempre numa SMT bem determinada T +
- A descrição dos diferentes estados é feita através de um conjunto de variáveis $\{x_i\}_{i=1}^{n}$ que tomam valores em T. Uma determinada combinação dos valores das x_i determina um estado específico. Assim o conjunto de variáveis determina completamente o espaço de estados.
  Em relação às FSM’s esta é uma diferença essencial; nas FSM os elementos do espaço de estados podem ser finitamente enumerados; nas FOTS são as variáveis que são finitamente enumeradas.

- A forma como são descritos os conjuntos de estados, em particular o conjunto dos estados iniciais. Nas FOTS um conjunto de estados é sempre representado por um predicado em T. Estes predicados são escritos como fórmulas
  $$f(x_1, \cdots, x_n)$$
  expressas nas variáveis x_i que definem o espaço de estados.

- A relação de transição $\delta \subseteq Q \times Q$ é também um conjunto e, por isso, deve também ser descrita por um predicado. Porém os elementos desse conjunto são pares $s \rightarrow s'$ que relacionam o estado “antes” da transição com o estado “depois” da transição; por isso o predicado deve ter dois argumentos.

Em termos de variáveis vamos ter de considerar um conjunto de variáveis "clone" das variáveis x_i para definir o estado pós-transição. Estas variáveis são designadas por $x_i^{'}$.
Desta forma a relação de transição é descrita por um predicado envolvendo ambos os conjuntos de variáveis.

$$\delta(x_1, \cdots, x_n, x_1^{'}, \cdots, x_n^{'})$$

Os valores das variáveis x_i determinam o estado antes da transição enquanto que os valores das $x_i^{'}$ determinam-no após a transição.

Exemplo (C)

Antes de considerarmos uma definição formal de um FOTS vamos aplicar estes princípios ao Exemplo (A) atrás estudado. Vimos que o programa imperativo aí apresentado

```
assert (z >= 0)
0: while z > 0:
1:     z = z - 1
2: stop
```

não podia ser descrito corretamente por uma FSM. Vamos ver como descrevê-lo por um FOTS.

1. O primeiro passo é a escolha de uma SMT apropriada. Neste exemplo, uma vez que apenas são manipulados números inteiros, essa SMT pode ser a LIA.

2. O espaço de estados é determinado por duas variáveis, ambas do tipo `Int` : a variável de iteração z e o "program counter" c. Implícitas existem também as suas cópias "próximo estado" $z^{'}$ e $c^{'}$.

3. O conjunto dos estados iniciais é determinado pelo predicado

$$\mathbf{init}(z, c) \equiv (c = 0) \wedge (z \geq 0)$$

4. A relação de transição é definida pelo predicado

$$\mathbf{trans}(z, c, z', c') \equiv \begin{cases} (c = 0) \wedge (z = 0) \wedge (c' = 2) \wedge (z' = z) & \vee \\ (c = 0) \wedge (z > 0) \wedge (c' = 1) \wedge (z' = z) & \vee \\ (c = 1) \wedge (z > 0) \wedge (c' = 0) \wedge (z' = z - 1) & \vee \\ (c = 2) \wedge (c' = 2) \wedge (z' = z) \end{cases}$$

Resumidamente

> *Uma FOTS é um tuplo* $\langle \mathrm{T}, \mathrm{X}, \mathbf{init}, \mathbf{trans} \rangle$ *em que* T *é uma teoria de base SMT,* X *é um vetor de variáveis com domínio em* T *,* $\mathbf{init}(\mathrm{X})$ *é um predicado que descreve o conjunto dos estados iniciais e* $\mathbf{trans}(\mathrm{X}, \mathrm{X}')$ *é um predicado que descreve a relação de transição.*

BTL (de novo)

Para descrever as propriedades lógicas de um FOTS vamos continuar a usar o BTL mas

- não são necessários "labels": as proposições atómicas são os predicados da teoria T .
- não é necessária uma função de "labelling" porque a semântica das proposições atómicas está completamente determinada pela teoria T ,
- a semântica do operador "until" só está defindа para traços limitados, mesmo que sejam infinitos.

A variante do BTL usada nos FOTS é definida de forma análoga à atrás referida

$$\phi, \varphi \ ::= \ p \mid \phi \rightarrow \varphi \mid \neg\phi \mid \mathrm{X}\phi \mid \phi \,\mathrm{U}\, \varphi$$

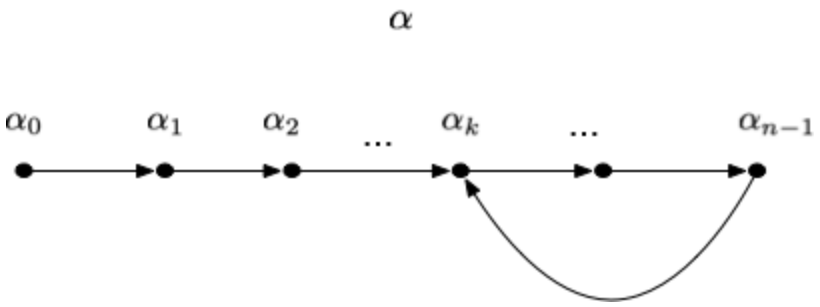
sendo p um predicado na teoria T .

Antes de definirmos a semântica desta lógica temos de analisar alguns resultados simples relacionados com os traços infinitos e limitados.

1. Os traços limitados e infinitos α têm um número finito n de estados distintos e, como têm um "loop" no último estado, existe um estado α_k , com $k < n$, e uma

transição $\alpha_{n-1} \rightarrow \alpha_k$.
Estes traços limitados formam a classe designada por (n, k).

Quando $k = n - 1$ diz-se que o traço tem um "laço". A classe de todos tais traços representa-se por (n).



Um traço infinito e limitado α da classe (n, k)

2. Tendo em atenção que o traço é infinito mas é limitado, os estados $\alpha_0, \cdots, \alpha_{n-1}$ determinam todos os restantes estados do traço. De facto
   i. se $k = n - 1$ então $\alpha_i = \alpha_{n-1}$, para todo $i \geq n$,
   ii. se $k < n - 1$ então $\alpha_{i+k} = \alpha_{j+k}$ quando $i \equiv j \bmod (n - k)$

3. Representamos por α' o sufixo de α que se inicia no estado α_1. Genéricamente representamos por $\alpha^{(i)}$ o sufixo de α que se inicia no estado α_i.
$$\alpha^{(0)} \equiv \alpha \quad , \quad \alpha^{(i+1)} \equiv (\alpha^{(i)})'$$

É agora possível definir a semântica de uma qualquer fórmula em BTL num traço infinito e limitado α:

1. $\alpha \models p$ sse $p(\alpha_0)$ é válido na teoria T

2. $\alpha \models \neg\phi$ e $\alpha \models \phi \rightarrow \varphi$ definem-se como na lógica proposicional.

3. $\alpha \models \mathrm{X}\,\phi$ sse $\alpha' \models \phi$

4. $\alpha \models \phi \,\mathrm{U}\, \varphi$ sse $\exists\, j < n \cdot \alpha^{(j)} \models \varphi$ e $\forall\, i < j \cdot \alpha^{(i)} \models \phi$

A semântica dos operadores F e G pode ser obtida a partir da semântica do operador "until" mas também pode ser expressa diretamente

i. $\alpha \models \mathrm{F}\,\phi$ sse $\exists\, i < n \cdot \alpha^{(i)} \models \phi$
ii. $\alpha \models \mathrm{G}\,\phi$ sse $\forall\, i < n \cdot \alpha^{(i)} \models \phi$

Em muitos casos os operadores U, F e G são aplicados a predicados p, q; nesses casos a semântica desses operadores pode ser escrita de forma mais simples:

i. $\alpha \models p\,\mathrm{U}\,q$ sse $\exists\, j < n \cdot q(\alpha_j)$ e $\forall\, i < j \cdot p(\alpha_i)$
ii. $\alpha \models \mathrm{F}\,p$ sse $\exists\, j < n \cdot p(\alpha_j)$
iii. $\alpha \models \mathrm{G}\,p$ sse $\forall\, i < n \cdot p(\alpha_i)$

"Bounded Model Checking" (BMC)

No contexto de um FOTS

$$\Sigma \equiv \langle \mathrm{T}, \mathrm{X}, \mathrm{init}, \mathrm{trans} \rangle$$

a verificação da validade semântica de uma fórmula temporal ϕ no comportamento de Σ, representado por

$$\Sigma \models \phi$$

consiste em decidir se, em <u>todos os traços</u> $\alpha \in \Sigma$, é válido

$$\alpha \models \phi$$

A estratégia genérica para resolver este problema tem quatro partes,

---

1. Codificar um traço α arbitrário definido uma variável α_i para cada estado no traço.
2. Codificar, numa fórmula de T, a relação $\alpha \in \Sigma$. Para isso é necessário codificar que
   i. verifica-se $\alpha_i \rightarrow \alpha_{i+1}$ para todos os estados do traço; esta condição exprime-se pela validade da conjunção $\bigwedge_i \mathrm{trans}(\alpha_i, \alpha_{i+1})$.
   ii. α_0, o primeiro estado de α, é um estado inicial; i.e. $\mathrm{init}(\alpha_0)$ tem de ser válido.
3. Codificar numa fórmula de T a relação $\alpha \models \phi$ usando as regras que definem a semântica da lógica temporal usada.
   Por exemplo, se p for um predicado e se for $\phi \equiv \mathrm{F}\,p$, então teremos $\alpha \models \phi$ codificado como $\bigvee_i p(\alpha_i)$.

4. Construir a "query"

$$(\alpha \in \Sigma) \wedge \neg(\alpha \vDash \phi)$$

e submetê-la ao "solver" SAT .

Se o SAT devolver a mensagem **unsat** então concluí-mos que a implicação semântica

$$(\alpha \in \Sigma) \Rightarrow (\alpha \vDash \phi)$$

se verifica: não existe nenhum traço α de Σ que não verifique $\alpha \vDash \phi$.
Se o SAT devolver a mensagem **sat** concluímos que existe pelo menos um traço α onde ϕ não se verifica. Tais traços são designados por **contra-exemplos**.

O **BMC** é uma aproximação à solução deste problema restringindo-o de duas formas:

- as fórmulas ϕ pertencem à "bounded temporal logic", e
- os traços α restringem-se aos traços limitados.

Nomeadamente cada traço BMC é visto dentro de uma classe $\Sigma_{(n,k)}$ formada por todos os traços com n estados distintos e com um "loop" para a posição k. Por isso, em rigor, a metodologia **BMC** <u>não</u> decide sobre a validade da relação genérica $\Sigma \vDash \phi$ mas decide sobre a validade de

$$\Sigma_{(n,k)} \vDash \phi$$

A codificação deste problema num problema SAT em T processa-se no seguinte processo:

---

1. Um traço limitado genérico α da classe (n, k) é representado por uma sequência finita de variáveis $\alpha \equiv (\alpha_0, \cdots, \alpha_{n-1})$ cada uma das quais é uma cópia das variáveis X no FOTS.

2. A relação $\alpha \in \Sigma_{(n,k)}$ é descrita por uma fórmula de T que contém a seguinte informação
   i. α_0 é um estado inicial do FOTS
   ii. para todo $0 < i < n$ existe uma transição $\alpha_{i-1} \rightarrow \alpha_i$ no FOTS
   iii. existe a transição que fecha o "loop", $\alpha_{n-1} \rightarrow \alpha_k$

Ou seja

$$\alpha \in \Sigma_{(n,k)} \equiv \text{init}(\alpha_0) \wedge \bigwedge_{i=1}^{n-1} \text{trans}(\alpha_{i-1}, \alpha_i) \wedge \text{trans}(\alpha_{n-1}, \alpha_k)$$

3. Usando a semântica da BTL codifica-se a relação $\alpha \models \phi$, num predicado em T.

4. Usa-se o SAT com a fórmula

$$\Phi \equiv (\alpha \in \Sigma_{(n,k)}) \wedge \neg(\alpha \models \phi)$$

Se esta fórmula for inconsistente então fica verificada a validade semântica de ϕ no sub-comportamento $\Sigma_{(n,k)} \subseteq \Sigma$ formado por todos os traços da classe (n,k).
Se Φ for consistente, então ϕ não é semanticamente válido e uma testemunha α é um contra-exemplo.

Note-se que

Se a fórmula Φ for consistente no comportamento $\Sigma_{(n,k)}$, qualquer testemunha dessa consistência, é um **contra-exemplo** que refuta a validade semântica de ϕ.
Nesse caso a fórmula ϕ nunca é semanticamente válida mesmo no comportamento total Σ.

Porém, o facto da condição $\alpha \models \phi$ se verificar para todos os traços da classe (n,k), não impede que exista um eventual traço $\alpha \in \Sigma$ que não pertence ao comportamento $\Sigma_{(n,k)}$ onde a condição não se verifica.
Portanto, se se provar a inconsistência de Φ no comportamento limitado $\Sigma_{(n,k)}$, não se pode garantir, com rigor, a inconsistência de Φ no comportamento Σ.

Acresce ainda que nem mesmo a inconsistência de Φ para <u>todos os traços limitados</u> pode, em rigor, ser testada pelo algoritmo acima. De facto um traço limitado é codificado por uma sequência de variáveis com tamanho finito. No entanto esse tamanho não é limitado e, ao invés, o algoritmo necessita de declarar um número limitado de variáveis n e uma posição de "loop" k

Isto sugere que o algoritmo deve definir uma estratégia para testar valores crescentes de n até que para exista confiança suficiente num resultado de inconsistência.

Exemplo (D)

Vamos considerar de novo o programa que temos vindo a considerar nos restantes exemplos deste capítulo. O programa é

```
assert (z >= 0)
0: while z > 0:
1:     z = z - 1
2: stop
```

e no exemplo (C) construímos o FOTS que o descreve.

Nesse FOTS o espaço de estados é determinado por pares (z, c). A variável α denota traços com n estados

$$\alpha \equiv (\alpha_0 , \alpha_1 , \cdots , \alpha_{n-1})$$

e cada α_i é um par de variáveis inteiras $\alpha_i \equiv (z_i, c_i)$. Vamos supor, por hipótese, que existe um laço no último estado de α: o estado α_n coincide com α_{n-1}.

A condição $\alpha \in \Sigma_{(n)}$ codifica-se no predicado

$$\text{init}(z_0, c_0) \wedge \bigwedge_{i=0}^{n-1} \text{trans}(z_i, c_i, z_{i+1}, c_{i+1})$$

A propriedade que vai ser verificada neste sistema dinâmica, tal como nos exemplos (A) e (B), vai afirmar que algures no futuro o programa termina. Pode-se escrever como

$$\phi \equiv \quad \mathrm{F}\,(c = 2)$$

A sua negação será então $\neg\phi \equiv \mathrm{G}\,(c = 0) \vee (c = 1)$, cuja codificação num predicado será

$$\bigwedge_{i=0}^{n-1} ((c_i = 0) \vee (c_i = 1))$$

> Para evitar problemas de decidibilidade evitamos usar o predicado $(c \neq 2)$. Neste problema, onde se tem sempre $c \in \{0, 1, 2\}$, o predicado $(c \neq 2)$ pode ser substituído por $(c = 0) \vee (c = 1)$.

Resumindo, o predicado cuja consistência devemos verificar é

$$\Phi_{(n)} \equiv \text{init}(z_0, c_0) \wedge \bigwedge_{i=0}^{n-1} \text{trans}(z_i, c_i, z_{i+1}, c_{i+1}) \wedge ((c_i = 0) \vee (c_i = 1))$$

Safety First Order Transition Systems (SFOTS)

Em vez de tentar verificar num FOTS uma propriedade arbitrária escrita numa lógica temporal, como o BTL, muitos problemas de verificação limitam-se à verificação de *propriedades de segurança* do tipo

> *"algo indesejável nunca ocorre".*

Nestes sistema o "algo indesejável" exprime-se através de um **predicado de erro** E que se adiona ao FOTS. Assim, num contexto determinado pela teoria base e as variáveis (T, X) que determina o espaço de estados, define-se um **safety FOTS** (ou **SFOTS)** como um triplo

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

definido pelo conjunto dos *estados iniciais* I, a *relação de transição* T e o conjunto dos estados de erro E.

Num tal SFOTS

- Um estado r é **acessível** ("**reachable**") se é um estado incial — i.e. $\mathrm{I}(r)$ é válido — ou então existe um outro estado acessível s tal que $\mathrm{T}(s, r)$ é válido.
- Um estado u é **"unsafe"** (**proibido**) se é um estado de erro — i.e. $\mathrm{E}(u)$ é válido — ou então existe um outro estado proibido v tal que $\mathrm{T}(u, v)$ (ou $\mathrm{T}^{-1}(v, u)$) é válido.
- O sistema Σ é **inseguro** quando existe um estado proibido que também é acessível.

Pela definição **o conjunto R dos estados acessíveis** ("reachable") é definido como o predicado mais fraco que verifica as seguintes condições

i. $\mathrm{I}(x) \rightarrow R(x)$ é taulológicamente válido
ii. $R(y) \wedge \mathrm{T}(y, x) \rightarrow R(x)$ é tautológicamente válido

Igualmente o **conjunto U dos estados inseguros** ("unsafe") é definido como o predicado mais fraco que verifica

i. $\mathrm{E}(x) \rightarrow U(x)$ é tautológicamente válido,

ii. $U(y) \wedge \mathrm{T}^{-1}(y,x) \rightarrow U(x)$ é tautológicamente válido

1. Uma relação binária $\phi(x,y)$ tem uma inversa que se obtém trocando a ordem dos argumentos

$$\phi^{-1}(x,y) \equiv \phi(y,x)$$

2. Recorde-se que, dadas duas relações binárias $\phi(x,y)$ e $\varphi(x,y)$, prova-se a condição

$\phi(x,y) \rightarrow \varphi(x,y)$ é tautológicamente válido

através de

$\phi(x,y) \wedge \neg\varphi(x,y)$ é unsat

Para provar que não existe nenhum estado que seja simultaneamente acessível e proibido basta provar que

$$R(x) \wedge U(x) \quad \text{é} \quad \text{unsat}$$

Se não for possível provar esta condição existirá um modelo para o predicado $R \wedge U$ que cada solução do SAT é um estado indicativos do erro (um estado acessível e proibido).

É evidente que esta metodologia só pode funcionar quando for possível calcular exatamente os predicados R e U. Só raramente é possível determinar estes predicados exatamente (por motivos que analisaremos no próximo capítulo) mas é possível computar limites superiores R^* e U^*; de facto, é sempre possível determinar predicados R^* e U^* tais que tanto $R \rightarrow R^*$ como $U \rightarrow U^*$ são tautologias.

Desta forma, se $R^* \wedge U^*$ for **unsat**, então necessariamente $R \wedge U$ também é **unsat:** se a solução aproximada garante segurança então o sistema é seguro. O inverso porém não é necessariamente verdadeiro: pode existir um contra-exemplo $s \in R^* \cap U^*$ que não pertença a $R \cap U$.

O conjunto R dos estados acessíveis e o conjunto U dos estados proibidos são definidos como os menores conjuntos de estados que verificam uma determinada relação de recorrência.

Nomeadamente definindo os conjuntos por predicados pode-se definir com rigor o conjunto R_k dos estados que são acessíveis de um estado inicial I em exatamente k passos da forma seguinte

$$\begin{cases} R_0(x) & \equiv & \mathrm{I}(x) \\ R_k(x) & \equiv & R_{k-1}(x) \ \vee \ (\exists\, y \cdot R_{k-1}(y) \ \wedge \ \mathrm{T}(y,x)) \qquad \text{para todo } k > 0 \end{cases}$$

O conjunto R dos estados acessíveis coincide com a união de todos os R_k

$$R \ \leftrightarrow \ \bigvee_k R_k$$

Alternativamente R pode ser visto como o menor conjunto que contém todos os R_k ; isto é, para todo $k \geq 0$, a fórmula $(R_k \ \rightarrow \ R)$ é tautológicamente válida.

De forma dual pode-se definir o conjunto dos estados inseguros ("unsafe") em m ou menos passos pela relação de recorrência

$$\begin{cases} U_0(x) & \equiv & \mathrm{E}(x) \\ U_m(x) & \equiv & U_{m-1}(x) \ \vee \ (\exists\, y \cdot U_{m-1}(y) \ \wedge \ \mathrm{T}^{-1}(y,x)) \qquad \text{para todo } m > 0 \end{cases}$$

De forma análoga o conjunto U de todos os estados "unsafe" verifica $U \ \leftrightarrow \ \bigvee_m U_m$ e U ainda pode ser interpretado como o predicado mais forte tal que $U_m \ \rightarrow \ U$ é tautológicamente válido para todo $m \geq 0$.

> *A condição de segurança do SFOTS* $\Sigma \equiv \langle \mathrm{I}, \mathrm{T}, \mathrm{E} \rangle$ *será então expressa como*
>
> $$R_k \ \wedge \ U_m \text{ é } \mathbf{unsat} \text{ para todos } k, m \geq 0$$

Verificação FOTS em traços não-limitados

Por vezes não é possível ou não é eficiente usar a metodologia BMC na verificação de propriedades dinâmicas de um FOTS; isso ocorre frequentemente quando, mesmo aumentando o número n de estados em cada traço, a sequência de comportamentos $\{\Sigma_{(n)}\}_{n>0}$ não converge de forma suficientemente rápida. Note-se n é o principal parâmetro que afecta a complexidade do algoritmo SAT.

Nestes casos não é possível assumir que o número de estados distintos é limitado. Como os traços não são limitados a sua descrição, para efeitos da verificação de propriedades, tem de usar algum processo indutivo.

Pelo mesmo motivo a semântica da lógica temporal não pode ser definida da mesma forma que foi feita para a "bounded temporal logic": as definições do significado dos operadores temporais $\mathrm{G}, \mathrm{F}\,\mathrm{e}\,\mathrm{U}$ são muito mais complexas e não permitem, no caso geral, uma formulação em SMT's.

Por essas razões, enquanto com o BMC é possível verificar qualquer fórmula ϕ da "bounded temporal logic", com traços não limitados é necessário restringir a classe de fórmulas que podem ser verificadas.
Normalmente é possível verificar a validade semântica das designadas *propriedades de segurança.* Isto *é,* fórmulas da forma $\mathrm{G}\,P$ em que P é um predicado,

Problema

- Dados um FOTS
$$\Sigma \equiv \langle \mathrm{P}, \mathrm{X}, \mathrm{init}, \mathrm{trans} \rangle$$
e um predicado P.
- Pretende-se verificar que, se α é um traço do comportamento de Σ, então P é válido em todos os estados desse traço. Isto é
$$\alpha \in \Sigma \;\Rightarrow\; \forall n \geq 0 \cdot P(\alpha_n)$$

O que significa $\alpha \in \Sigma$?
Recordemos que, se for $\alpha \equiv \langle \alpha_0, \cdots, \alpha_n, \alpha_{n+1}, \cdots \rangle$, e se se verificar $\alpha \in \Sigma$ então verifica-se necessariamente

i. é válido $\mathrm{init}(\alpha_0)$
ii. para todo $n \geq 0$ é válido $\mathrm{trans}(\alpha_n, \alpha_{n+1})$.

Isto é
$$\alpha \in \Sigma \Rightarrow (\mathrm{i}) \wedge (\mathrm{ii})$$

O que significa $\forall n \geq 0 \cdot P(\alpha_n)$?
O significado desta fórmula quantificada estabelece-se através do princípio da indução da aritmética. Nomeadamente verifica-se $\forall n \geq 0 \cdot P(\alpha_n)$ quando

iii. é válido $P(\alpha_0)$
iv. para todo $n \geq 0$ é válido $P(\alpha_n) \rightarrow P(\alpha_{n+1})$.

Aqui a implicação é

$$(\text{iii}) \ \wedge \ (\text{iv}) \ \Rightarrow \ \forall \, n \geq 0 \ \cdot \ P(\alpha_n)$$

Temos agora de juntar estes dois pares de condições. Por razões que serão evidentes em seguida vamos designar esta condição por 0-**Indução**

0-**Indução**

Para um traço $\alpha \in \Sigma$ *arbitrário, deve-se verificar as seguintes condiçõe*

(a) $\text{init}(\alpha_0) \rightarrow P(\alpha_0)$ *é válido*

(b) para todo n, $\text{trans}(\alpha_n, \alpha_{n+1}) \rightarrow (P(\alpha_n) \rightarrow P(\alpha_{n+1}))$ *é válido*

Nestas circunstâncias obtemos o resultado essencial desta metodologia

$$0\text{–Induction} \ \Rightarrow \ \forall \, n \ \cdot \ P(\alpha_n)$$

É essencial notar-se que este princípio não pode ser usadas no sentido inverso. Isto é, se uma ou várias destas condições (a) e (b) não se verificarem, não se pode concluir que $\forall \, n \cdot P(\alpha_n)$ também não se verifica.

Indução Simples

Normalmente aplicação direta da 0-**Indução** não é viável; tal implicaria conhecer especificamente os diversos estados α_n que, como sabemos, formam uma sequência infinita.

A *indução simples* , ou 1-**Indução** é uma aproximação às condições da 0-**Indução** substituindo nas condições (i) e (ii), a quantificação "α é um traço arbitrário" pela quantificação em relação a todos os pares de estados x, y quer esses estados pertençam a um traço ou não.

1-**Indução**

Para todo par de estados x, y *deve-se verificar*

(a) $\text{init}(x) \rightarrow P(x)$ *é tautologicamente válido*

(b) $\text{trans}(x, y) \rightarrow (P(x) \rightarrow P(y))$ *é tautológicamente válido*

Obviamente a quantificação "para todos os pares de estados x, y" inclui a quantificação "para todos os pares consecutivos α_n, α_{n+1} do traço α" , Portanto, se

as condições da 1-**Indução** se verificarem então também se verificam as condições da 0-**Indução** : a 1-**Indução** tem condições **mais fortes** do que as da 0-**Indução.**

$$1\text{-Induction} \Rightarrow 0\text{-Induction}$$

Também se as condições da 1-**Indução** não se verificarem isso não implica que as condições da 0-**Indução** não se verificam.

A grande vantagem computacional da 1-**Indução** em relação à 0-**Indução** é o facto de poderem ignorar traços específicos e poderem ser codificados como um problema SAT.
Para codificar estas duas condições em SAT temos de introduzir variáveis X e Y e usar as abordagem das provas por contradição

a. $\text{init}(X) \wedge \neg P(X)$ é **unsat**
b. $\text{trans}(X, Y) \wedge P(X) \wedge \neg P(Y)$ é **unsat**

É conveniente referir mais uma vez que o facto de qualquer destas aproximações não se verificar, o que ocorre quando o SAT devolve a mensagem **sat** , não implica que a condição original seja inválida. É esta incerteza que os restantes métodos de aproximação procuram diminuir.

Exemplo (E)

Considere-se o seguinte código

```
assert (a > 0) and (b > 0)
0: while (a != b):
     if a > b then a = a - b
     if a < b then b = b - a
1: stop
```

Pretende-se verificar a propriedade de segurança $\tau \equiv (a > 0) \wedge (b > 0)$.

Vamos começar por definir o FOTS que descreve este programa.

1. A teoria SMT é uma qualquer teoria base contento os inteiros; por exemplo, LIA.
2. O espaço de estados é definido por três variáveis inteiras: a, b e o "program counter" c.
   O domínio das duas primeiras é N, e o do "program counter" é binário $\{0, 1\}$.
3. O conjunto dos estados iniciais é determinado pelo predicado
$$\mathrm{init}(a, b, c) \equiv (a > 0) \wedge (b > 0) \wedge (c = 0)$$
4. A relação de transição é descrita pelo predicado
$$\mathrm{trans}(a, b, c, a', b', c') \equiv$$
$$\begin{cases} (a = b) \wedge (c = 0) \wedge (a' = a) \wedge (b' = b) \wedge (c' = 1) & \vee \\ (a > b) \wedge (c = 0) \wedge (a' + b = a) \wedge (b' = b) \wedge (c' = 0) & \vee \\ (a < b) \wedge (c = 0) \wedge (b' + a = b) \wedge (a' = a) \wedge (c' = 0) & \vee \\ (c = 1) \wedge (c' = 1) \wedge (a' = a) \wedge (b' = b) \end{cases}$$

Note-se que, principalmente quando são variáveis de domínio infinito, quase sempre é boa política evitar predicados $(a \neq b)$.
Neste programa só interessa realmente duas posiçoes do "program counter": no início do ciclo, representado por 0 e no final, representado por 1.

O uso do princípio de indução para verificar a propriedade $\mathrm{G}(a > 0) \wedge (b > 0)$ vai consistir em provar a inconsistência dos dois predicados seguintes

- $\mathrm{init}(a, b, c) \wedge ((a \leq 0) \vee (b \leq 0))$
- $\mathrm{trans}(a, b, c, a', b', c') \wedge (a > 0) \wedge (b > 0) \wedge ((a' \leq 0) \vee (b' \leq 0))$

---

k-Indução

$$\Sigma \equiv \langle X, \mathrm{init}, \mathrm{trans} \rangle$$

A k-**Indução** generaliza a 1-**Indução** quantificando não só em relação a pares de variáveis X, Y, usadas na indução simples , para um tuplo de $k+1$ variáveis representadas por

$$\mathrm{X}^{(0)}, \mathrm{X}^{(1)}, \cdots, \mathrm{X}^{(k)}$$

Na indução simples pode acontecer que o predicado P não permita provar a tautologia de

$$P(X) \ \wedge \ \text{trans}(X, Y) \ \rightarrow \ P(Y)$$

porque a hipótese $P(X) \ \wedge \ \text{trans}(X, Y)$ é um antecedente demasiado fraco para forçar a conclusão $P(Y)$ tautologicamente .

Uma possível solução passa por reforçar o conjunto de hipóteses. Por exemplo, pode-se introduzir um estado intermédio Z entre X e Y e exigir que nesse estado intermédio também se verifique P . Isto é, a condição (b) assume a forma

(b) $P(X) \wedge P(Z) \wedge \text{trans}(X, Z) \wedge \text{trans}(Z, Y) \ \rightarrow \ P(Y)$ é tautologia

No entanto a condição (a) também tem de ser substituída, neste caso, por duas condições: a primeira é equivalente à da indução simples: diz que num qualquer estado inicial a propriedade P tem de ser válida

(a1) $\text{init}(X) \ \rightarrow \ P(X)$ é tautologia

A segunda condição diz que num estado imediatamente a seguir a um estado inicial deve-se comportar como um estado inicial: o predicado P também tem de ser válido

(a2) $\text{init}(X) \ \wedge \ \text{trans}(X, Y) \ \rightarrow \ P(Y)$ é tautologia

Esta tecnica de prova designa-se por **"2-induction"**.

Como a quantificação "para todos X, Y, Z" inclui a quantificação "para todos X, Y" , sendo válidas as condições da 2-induction são válias as condições da 1-induction.

No contexto dos FOTS, seguindo os mesmo passos que se usou na indução simples, a 2**-induction** vai exigir predicados em 3 variáveis: X , Y e Z.

Dado um predicado P , para provar que $\text{G}P$ é semânticamente válido no FOTS Σ é suficiente provar que são inconsistentes todos os predicados

(a1) $\text{init}(X) \ \wedge \ \neg P(X)$

(a2) $\text{init}(X) \wedge \text{trans}(X, Y) \wedge \neg P(Y)$

(b) $P(X) \wedge \text{trans}(X, Y) \wedge P(Y) \wedge \text{trans}(Y, Z) \wedge \neg P(Z)$

Generalizando, a k-**induction** vai exigir uma formulação em $k+1$ variáveis: $X^{(0)}, \cdots, X^{(k)}$.

Dado um predicado P, para provar que $\mathrm{G}\,P$ é semânticamente válido no FOTS Σ é suficiente provar que são inconsistentes todos os predicados

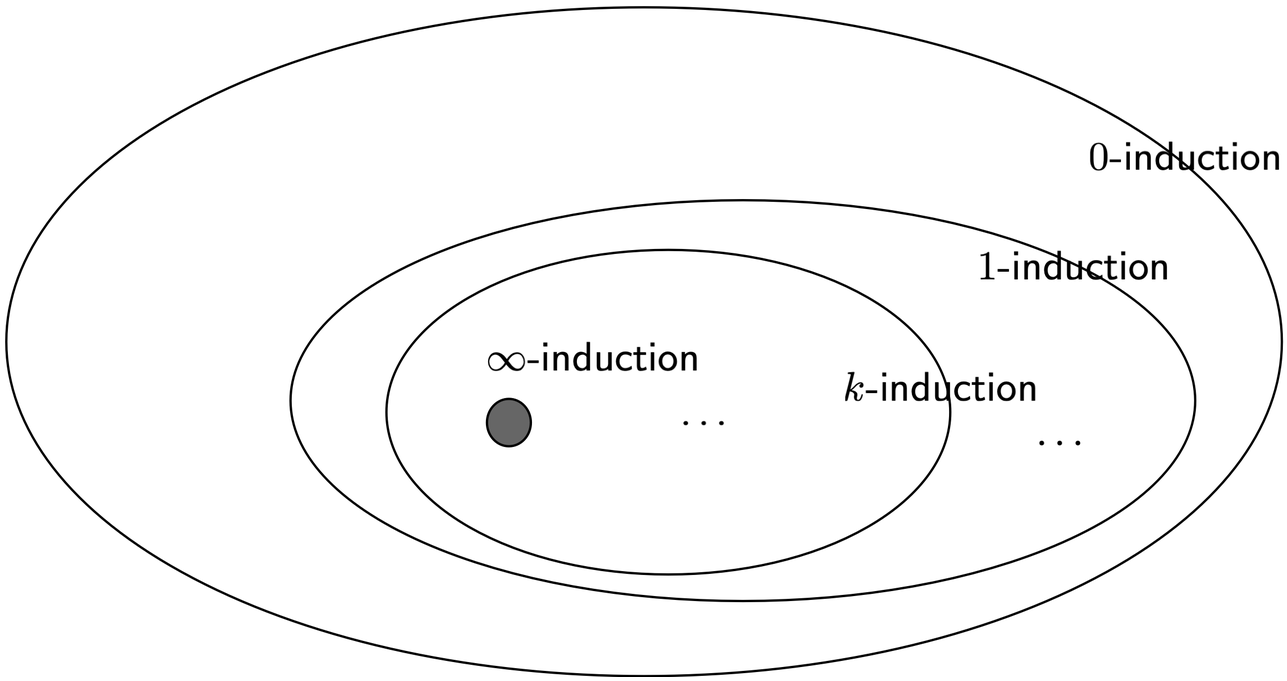
(a) $\text{init}(X^{(0)}) \wedge \bigwedge_{i=0}^{j-1} \text{trans}(X^{(i)}, X^{(i+1)}) \wedge \neg P(X^{(j)})$
para todo $j = 0, \cdots, k-1$

(b) $(\bigwedge_{i=0}^{k-1} P(X^{(i)}) \wedge \text{trans}(X^{(i)}, X^{(i+1)})) \wedge \neg P(X^{(k)})$

Para provar a inconsistência procedemos da forma usual: submete-se cada um dos predicados como "query" de um SAT e verifica-se a resposta do "solver".
Se a resposta for é **unsat,** então a condição em causa verifica-se. Ao contrário se a resposta for **sat** então a condição não se verifica e os modelos fornecidos pelo "solver" são contra-exemplos a partir dos quais será possível ver a razão dessa falha.

Quando maior for k mais provável é assegurar este as condições da k-**induction**. Nomeadamente provar as condições ∞-induction implicaria (se fosse implementado) incluir um número infinito de variáveis $X^{(i)}$ e ter apenas a condição (a) imposta. É semelhante a um BMC mas com comprimento infinito.

No entanto o número de variáveis a incluir na prova cresce linearmente com k o que aumenta o espaço de estados e, por isso, a complexidade das provas.



Em resumo a relação entre estas coleções de condições será sempre

$$\infty\text{-induction} \cdots \Rightarrow k\text{-induction} \Rightarrow \cdots \Rightarrow 1\text{-induction} \Rightarrow 0\text{-induction} \Rightarrow \mathrm{G}\,P$$

Considerações genéricas sobre indução

Limites superiores indutivos

No estudo da verificação de propriedades $\mathrm{G\,P}$ (as "boas propriedades universais") num SFOTS

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

pode-se considerar que a "boa propriedade" P determina o predicado de erro

$$\mathrm{E} \equiv \neg P$$

Na discussão que se segue podemos estender toda análise ignorando as particularidades da propriedade P e considerando apenas as interações dos predicados I e E com as conjuntos

- $\mathrm{R} \equiv \bigvee_{i \geq 0} R_i$ é o *conjunto de estados acessíveis*
- $\mathrm{U} \equiv \bigvee_{j \geq 0} U_j$ é o *conjuntos de estados proibidos*

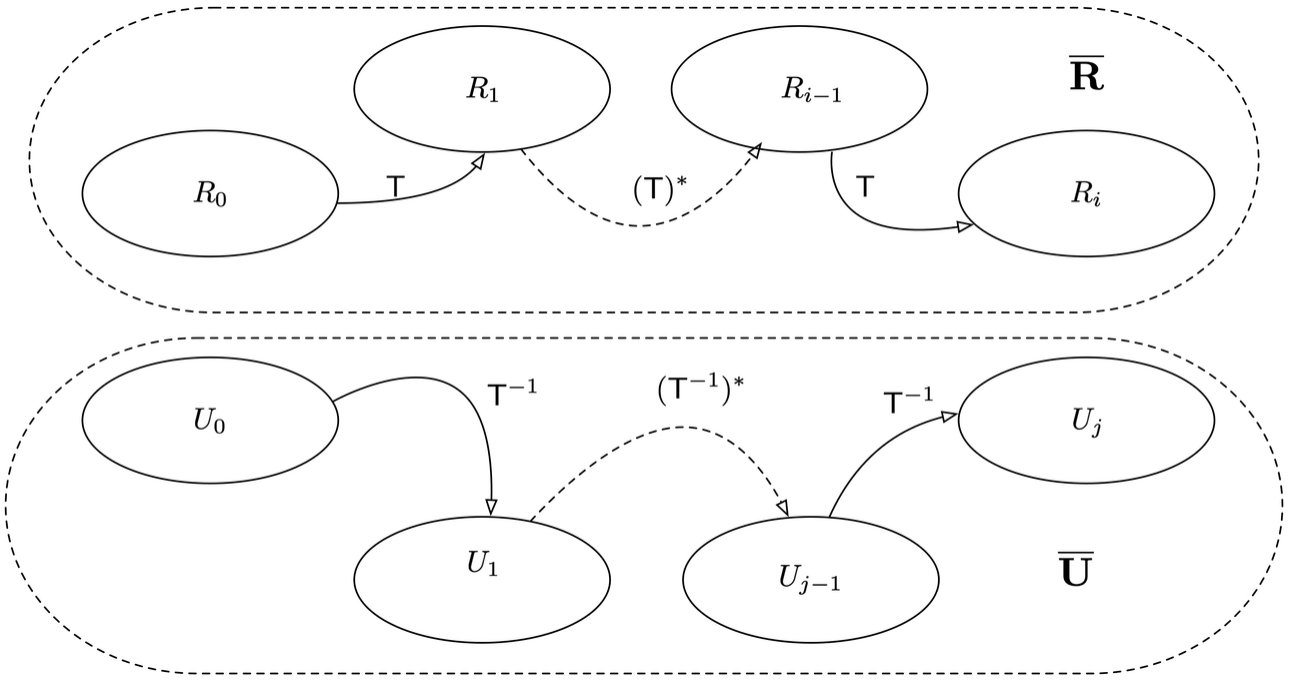
sendo $\{R_i\}$ a sequência dos conjuntos de estados acessíveis em exatamente i passos e $\{U_j\}$ a sequência dos conjuntos de estados proibidos em exatamente j passos.

Recorde-se que as sequências R_i e U_j são inicialisadas, normalmente, com I e E.

$$R_0 \equiv \mathrm{I} \quad , \quad U_0 \equiv \mathrm{E}$$

No entanto, por enquanto, vamos considerar que R_0 e U_0 são arbitrários e vamos apenas considerar as relações de recorrência que determinam os restantes elementos das sequências.

$$R_i(r) \equiv \exists\, s \cdot R_{i-1}(s) \wedge \mathrm{T}(s,r) \quad , \quad U_j(u) \equiv \exists\, v \cdot U_{j-1}(v) \wedge \mathrm{T}^{-1}(v,u)$$



No caso geral R e U podem não ser computáveis; isto é, pode não existir limites para a disjunções contáveis $\bigvee_i R_i$ e $\bigvee_j U_j$.

Sabemos, no entanto, que existe sempre um **limite superior,** que vamos representar por $\overline{R}$ para todos os R_k ; isto é um predicado tal que $R_i \rightarrow \overline{R}$ seja válido para todo i. Em último caso pode-se fazer $\overline{R} \equiv \mathrm{True}$.

Dualmente pode-se considerar um **limite superio**r $\overline{U}$ para todos os U_j ; isto é, pretende-se que seja $U_j \rightarrow \overline{U}$ para todo j .

Obviamente para garantir segurança usando estes limites superiores temos de garantir que $\overline{R}$ não intersecta $\overline{U}$. Por isso não se pode considerar limites superiores arbitrariamente grandes e é necessário verificar se existem limites superiores que possam garantir a segurança do SFOTS.

Dada uma relação de transição τ, um predicado φ é **indutivo para** τ quando verifica

$$\varphi(X) \wedge \tau(X, Y) \rightarrow \varphi(Y) \qquad \text{é tautologia}$$

> *Seja* $\{\phi_k\}_{k\geq 0}$ *é uma sequência de predicados que verificam, para todo* $k \geq 0$
> $$\phi_k(X) \wedge \tau(X, Y) \rightarrow \phi_{k+1}(Y) \qquad \text{é tautologia}$$
> *Se* φ *for indutivo e se for* $\phi_0 \subseteq \varphi$*, então verifica-se* $\phi_k \subseteq \varphi$ *para todo* $k \geq 0$

A prova é feita por indução: por hipótese a relação $\phi_k \subseteq \varphi$ verifica-se para $k = 0$. Se for $\phi_k \subseteq \varphi$ então $\phi_{k+1} = \tau(\phi_k) \subseteq \tau(\varphi) \subseteq \varphi$.

Este resultado permite dar condições suficientes para que $\overline{R}$ e $\overline{U}$ sejam limites superiores das sequências R e U respetivamente: basta que seja indutivos para a relação T e T^{-1} respetivamente e verifiquem $R_0 \equiv \mathrm{I}$ e $U_0 \equiv \mathrm{E}$.

“Look aheads” : induçao transfinita e propriedades de animação

Considere-se o seguinte algoritmo

```
assert type(z) == int and z > 0
0 : while z > 0
        z = z - 1
1 : assert z == 0
```

Quere-se provar que o algoritmo termina e termina corretamente usando verificação em traços infinitos; nomeadamente usando indução.

Ao contrário dos SFOTS’s este não é um algoritmo que é aferido por uma *propriedade de segurança* ; isto é, uma propriedade da forma $\mathrm{G}(\neg E)$ — “sempre no

futuro não se atinge um estado de erro". Estes são os problemas que usualmente são resolvidos por indução; de facto as técnicas de indução simples e k-indução, estão definidas para exatamente este tipo de propriedades.

Não é o caso deste problema. Pelo contrário aqui a propriedade que se pretende validar é uma *propriedade de animação*; isto é, uma propriedade da forma $\mathrm{F}\,P$ — "eventualmente no futuro atinge-se um estado de paragem".

Aqui em vez do predicado E determinando os "estados de erro", temos um predicado P que determina os "estados de paragem". Pode-se escrever $\mathrm{F}\,P \equiv \neg\,\mathrm{G}\,(\neg\,P)$ mas, de qualquer forma, continua a não ser evidente como se pode usar a indução neste caso.

A menos que exista algum predicado $P*$ tal que a validade de $\mathrm{G}\,P*$ force a validade de $\mathrm{F}\,P$. Se for possível construir $P*$, então pode-se usar indução para provar $\mathrm{G}\,P*$ e daí concluir a validade de $\mathrm{F}\,P$.

Por razões que analisaremos em seguida um tal predicado, se existir, será designado por **look ahead**. Intuitivamente é uma propriedade universal ($\mathrm{G}\,P*$ sugere-se que a propriedade é sempre válida) e por isso "olha à frente" do predicado P e de um eventual estado onde a propriedade deva ser válida.

Infelizmente, para um predicado arbitrário P, este tipo de construção nem sempre é possível. No entanto em certas situações particulares é possível. Antes de examinarmos, neste particular problema, como se faz a construção $P \rightsquigarrow P*$ convém construir o respetivo FOTS.

O FOTS que descreve este programa é definido por

- Um estado definido por um par de variáveis inteiras (c, z) com o contador $c \in \{0, 1\}$ e a variável $z \in \mathrm{Z}$. Estas variáveis determinam que a lógica SMT usada é a `QF_LIA`.
- O predicado definindo os estados iniciais $\mathrm{I} \equiv (c = 0) \wedge (z > 0)$
- Neste modelo acrescenta-se o predicado "estados de paragem" $\mathrm{P} \equiv (c = 1) \wedge (z = 0)$.
- A relação de transição é

$$\mathrm{T}(c, z, c', z') \equiv$$
$$(c = 0) \wedge (z > 0) \wedge (c' = 0) \wedge (z' = z - 1) \vee$$

$$(c = 0) \wedge (z = 0) \wedge (c' = 1) \wedge (z' = z) \vee$$
$$(c = 1) \wedge (c' = c) \wedge (z' = z)$$

Um passo intermédio para verificar que P é um estado de paragem legítimo consistirá em verificar se, uma vez atingido um estado $s \in \mathrm{P}$, toda a transição $\mathrm{T}(s, r)$ conduz a um estado r que ainda é um estado de paragem.
Por outras palavras, é necessário provar que P é um predicado indutivo o que pode ser verificado facilmente usando indução. Basta verificar se

$$\mathrm{P}(c, z) \wedge \mathrm{T}(c, z, c', z') \wedge \neg \mathrm{P}(c', z') \quad \text{é unsat}$$

A outra questão é saber como realmente se chega a esse primeiro estado $s \in \mathrm{P}$ a partir de um estado inicial $i \in \mathrm{I}$. Aqui temos de entrar com a particularidade de o programa frisar que a variável z tomar valores inteiros positivos que inicialmente verificam $z > 0$ e que, em cada passo até atingir $z = 0$, são decrementados uma unidade.
Neste caso o facto da relação $<$ nos números naturais N ser uma ordem bem fundada diz-nos que qualquer sequência estritamente decrescente de números naturais eventualmente termina em um mínimo. Nomeadamente não existem sequências infinitas de números naturais que sejam estritamente decrescentes. Como neste exemplo os valores de z não-nulos formam uma sequência estritamente decrescente, o mínimo tem de ser forçosamente zero.

Por isso, basta assegurar que em cada transição o valor de z é zero e permanece zero ou então é não-nulo e decresce.
Formalmente: definindo o predicado

$$P^*(c, z, c', z') \equiv (z > 0) \wedge (z' < z) \vee (z = 0) \wedge (z' = z)$$

o facto de os inteiros positivos formarem uma ordem bem fundada assegura que

$$\mathrm{G}\, P^* \rightarrow \mathrm{F}\,(z = 0) \qquad \text{é válido}$$

Para assegurar que $\mathrm{G}\,\mathrm{P}^*$ é válido usando indução (e é esse o nosso objectivo) temos que:

- garantir que P^* é indutivo, e
- nos estados iniciais P^* é válido

Tecnicamente temos uma dificuldade com este requisito uma vez que P^* não é, tal como os predicados $\mathrm{I}, \mathrm{E} \ldots$, um predicado unário que toma argumento um só

estado; pelo contrário, é um predicado binário que, tal como a relação de transição T, toma como argumentos dois estados: o antecessor e sucessor da transição.

Para ver o que significa um predicado binário ser indutivo vamos adiar por momentos este exemplo e vamos considerar o caso geral de um FOTS $\Sigma = \langle X, \mathrm{I}, \mathrm{T}, \mathrm{P} \rangle$ com variáveis de estado X, estados iniciais I, relação de transição T e estados de paragem P.

Considere-se um predicado binário genérico $\varphi(X, Y)$ que estabelece uma relação entre o valor das variáveis X e valores de variáveis Y. Quando se constrói a conjunção

$$\varphi(X, Y) \wedge \mathrm{T}(X, Y)$$

restringe-se a relação entre variáveis cujos valores determinam o estado antes e depois de uma transição. Porém se quisermos avaliar a evolução de $\varphi(X, Y)$ quando ambos os seus argumentos evoluem um passo de transição, então é necessário introduzir uma terceira variável Z que representam os valores de Y após uma transição; isto é, precisamos de considerar as consequências da conjunção

$$\varphi(X, Y) \wedge \mathrm{T}(X, Y) \wedge \mathrm{T}(Y, Z)$$

O predicado φ é indutivo quando a validade desta conjunção de predicados implicar a validade de $\varphi(Y, Z)$. Resumidamente

> *O predicado binário φ é indutivo quando*
>
> $$\varphi(X, Y) \wedge \mathrm{T}(X, Y) \wedge \mathrm{T}(Y, Z) \wedge \neg\varphi(Y, Z) \quad \text{é} \quad \textbf{unsat}$$

Para verificar com indução a validade de uma condição de segurança da forma $\mathrm{G}\,\varphi$ é necessário verificar a condição anterior para assegurar que φ é indutivo mas também é necessário verificar que φ é válido nos estados iniciais. Como φ depende do estado inicial mas também de um estado imediatamente seguinte, é necessário provar que

$$\mathrm{I}(X) \wedge \mathrm{T}(X, X') \rightarrow \varphi(X, X') \quad \text{é tautologia}$$

Regressando ao nosso exemplo, para verificar a validade de $\mathrm{G}\,\mathrm{P}^*$ temos de garantir que P^* é indutivo mas também que P^* é válido no estado inicial e num estado que se siga. Para isso temos de garantir

$$\mathrm{I}(c, z) \wedge \mathrm{T}(c, z, c', z') \wedge \neg\mathrm{P}^*(c, z, c', z') \quad \text{é} \quad \text{unsat}$$

Em resumo

Dado um FOTS $\Sigma = \langle X, \mathrm{I}, \mathrm{T}, \mathrm{P} \rangle$, associado a um conjunto P de *estados de paragem,* quere-se verificar

$$\Sigma \models \mathrm{F}\,\mathrm{P}$$

Segue-se a seguinte metodologia:

1. Em primeiro lugar é necessário verificar que realmente P é um conjunto de estados de paragem: em qualquer traço de Σ quando se alcança um estado onde P é válido, em todos os estados subsequentes P continuará válido.
   Para formalizar esta condição basta exigir que P seja um predicado indutivo; isto é, se Y denota um "clone" de X, então
   $$\mathrm{P}(X) \wedge \mathrm{T}(X, Y) \wedge \neg \mathrm{P}(Y) \quad \text{é} \quad \text{unsat}$$

2. O segundo passo consiste em encontrar (se possível) um predicado P^* que verifique
   $$\Sigma \models \mathrm{G}\,P^* \quad \text{implica} \quad \Sigma \models \mathrm{F}\,\mathrm{P}$$
   Esta condição não é, usualmente, verificável via SAT. Exige propriedades algébrica da teoria SMT de suporte tal como a invocada no nosso exemplo: o facto de a ordem nos inteiros ser bem-fundada.

3. Em terceiro lugar, o predicado P^* vai ter necessidade de "ver para além do estado presente" um determinado número de passos de transição. Seja k esse número de passos necessários. Então vão ser necessários $k+1$ "clones" das variáveis X aqui enumerados como $X_0, X_1, \cdots, X_k$ e o predicado P^* vai ter k argumentos; será $P^*(X_0, X_1, \cdots, X_{k-1})$.
   Como se pretende que P^* seja universal, vai-se exigir que o predicado seja indutivo. Essa exigência vai introduzir a 2ª condição de verificação
   $$P^*(X_0, \cdots, X_{k-1}) \wedge \bigwedge_{i=1}^{k} \mathrm{T}(X_{i-1}, X_i) \wedge \neg P^*(X_1, \cdots, X_k) \quad \text{é} \quad \text{unsat}$$

4. No entanto, não basta que P^* seja indutivo. Para verificar $\Sigma \models \mathrm{G}\,P^*$, basta verificar que no estado inicial de um traço arbitrário o predicado P^* é válido. Daqui resulta a última condição a verificar
   $$\mathrm{I}(X_0) \wedge \bigwedge_{i=1}^{k-1} \mathrm{T}(X_{i-1}, X_i) \wedge \neg P^*(X_0, \cdots, X_{k-1}) \quad \text{é} \quad \text{unsat}$$