Réécriture : un outil pour la vérification

Yohan Boichut

Cours Master IRAD – Semestre 3

Outline

Les termes

- $\mathcal{F}$: ensemble des symboles fonctionnels
- $\mathcal{X}$: ensemble des variables
- $\mathcal{T}(\mathcal{F})$: ensemble des termes clos
- $\mathcal{T}(\mathcal{F}, \mathcal{X})$ ensemble des termes ouverts (termes contenant des variables)

Exemple

Soient $\mathcal{F} : \{f : 2, a : 0, b : 0\}$ *et* $\mathcal{X} = \{x, y\}$. $f(a, b) \in \mathcal{T}(\mathcal{F})$ *et* $f(x, f(a, y)) \in \mathcal{T}(\mathcal{F}, \mathcal{X})$.

Les positions

- Une position ω est un mot de $\mathbb{N}^*$
- ϵ, le mot vide, est la position racine i.e. le sommet du terme
- $\epsilon.1 (= 1)$ correspond à la position du premier fils
- $\epsilon.1.2 (12$ correspond à la position du second fils du premier fils

Les positions

- Une position ω est un mot de $\mathbb{N}^*$
- ϵ, le mot vide, est la position racine i.e. le sommet du terme
- $\epsilon.1 (= 1)$ correspond à la position du premier fils
- $\epsilon.1.2 (12$ correspond à la position du second fils du premier fils

Exercice

Soient $\mathcal{F} : \{f : 2, a : 0, b : 0\}$ *et* $\mathcal{X} = \{x, y\}$.
Dans $f(x, f(a, y)) \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, *quelles sont les positions de* x, a *et* $f(a, y)$ *?*

Les positions

Définition

Soit t un terme de $\mathcal{T}(\mathcal{F}, \mathcal{X})$. L'ensemble des positions de t est noté $\mathcal{P}os(t)$ et est défini inductivement sur la structure de t comme suit :

- $\mathcal{P}os(t) = \epsilon$ *si* $t \in \mathcal{X}$
- $\mathcal{P}os(f(t_1, \ldots, t_n)) = \{\epsilon\} \cup \{i.p \mid 1 \leq i \leq n \wedge p \in \mathcal{P}os(t_i)\}$

Les positions

Définition

Soit t un terme de $\mathcal{T}(\mathcal{F}, \mathcal{X})$. L'ensemble des positions de t est noté $\mathcal{P}os(t)$ et est défini inductivement sur la structure de t comme suit :

- $\mathcal{P}os(t) = \epsilon$ *si* $t \in \mathcal{X}$
- $\mathcal{P}os(f(t_1, \ldots, t_n)) = \{\epsilon\} \cup \{i.p \mid 1 \leq i \leq n \wedge p \in \mathcal{P}os(t_i)\}$

Exercice

Pour $t = f(a, f(x, f(a, b)))$, calculer $\mathcal{P}os(t)$.

Opérations sur les termes

Soient $\mathcal{F}$ et $\mathcal{X}$. Pour $t, t' \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, et $\omega \in \mathcal{P}os(t)$,

- $t(\omega)$ retourne le symbole fonctionnel de t à ma position ω
- $t|_\omega$ retourne le sous-terme de t à la position ω
- $t[t']_\omega$ remplace le sous-terme de t à la position ω par t'
- $\mathcal{V}ar(t)$ retourne l'ensemble des variables apparaissant dans t

Opérations sur les termes

Soient $\mathcal{F}$ et $\mathcal{X}$. Pour $t, t' \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, et $\omega \in \mathcal{P}os(t)$,

- $t(\omega)$ retourne le symbole fonctionnel de t à ma position ω
- $t|_\omega$ retourne le sous-terme de t à la position ω
- $t[t']_\omega$ remplace le sous-terme de t à la position ω par t'
- $\mathcal{V}ar(t)$ retourne l'ensemble des variables apparaissant dans t

Exercice

Soient $t = f(a, f(b, g(x)))$, $t' = f(b, b)$ *et* $\omega = 22$. *Calculer*

- $t(\omega)$
- $t|_\omega$
- $t[t']_\omega$

Les substitutions

- Une substitution $\sigma : \mathcal{X} \mapsto \mathcal{T}(\mathcal{F})$ substitue des variables par des termes clos.
- Une substitution $\sigma : \mathcal{X} \mapsto \mathcal{T}(\mathcal{F}, \mathcal{X})$ substitue des variables par des termes ouverts.

Les substitutions

- Une substitution $\sigma : \mathcal{X} \mapsto \mathcal{T}(\mathcal{F})$ substitue des variables par des termes clos.
- Une substitution $\sigma : \mathcal{X} \mapsto \mathcal{T}(\mathcal{F}, \mathcal{X})$ substitue des variables par des termes ouverts.

Définition

Soit $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, l'application d'une substitution σ sur t se note $t\sigma$ et se définit de la façon suivante

- $t\sigma = \sigma(t)$ *si* $t \in \mathcal{X}$
- $f(t_1, \ldots, t_n)\sigma = f(t_1\sigma, \ldots, t_n\sigma)$

Les substitutions

- Une substitution $\sigma : \mathcal{X} \mapsto \mathcal{T}(\mathcal{F})$ substitue des variables par des termes clos.
- Une substitution $\sigma : \mathcal{X} \mapsto \mathcal{T}(\mathcal{F}, \mathcal{X})$ substitue des variables par des termes ouverts.

Définition

Soit $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, l'application d'une substitution σ sur t se note $t\sigma$ et se définit de la façon suivante

- $t\sigma = \sigma(t)$ *si* $t \in \mathcal{X}$
- $f(t_1, \ldots, t_n)\sigma = f(t_1\sigma, \ldots, t_n\sigma)$

Exercice

Soit $\sigma = \{x \mapsto g(a, y), y \mapsto b\}$. Appliquer σ au terme $f(f(a, y), g(x))$.

Maintenant nous avons tout pour faire de la réécriture !

Outline

La réécriture

Définition

La réécriture permet d'écrire quelque chose en quelque chose d'autre, d'où le nom.

La réécriture

Définition

La réécriture permet d'écrire quelque chose en quelque chose d'autre, d'où le nom.

Plus sérieusement, nous utilisons cette technique sur des structures comme

- Mots : un mot est *réécrit* en un autre mot
- Termes : un terme est *réécrit* en un autre terme

La réécriture

Pour définir *qui doit être réécrit en quoi*, nous parlons de règles de réécriture.

La réécriture

Pour définir *qui doit être réécrit en quoi*, nous parlons de règles de réécriture.

Exemple

- *Mots : $ab \longrightarrow ba$ ou $ax \longrightarrow x$ avec x une variable*

La réécriture

Pour définir *qui doit être réécrit en quoi*, nous parlons de règles de réécriture.

Exemple

- *Mots :* $ab \longrightarrow ba$ *ou* $ax \longrightarrow x$ *avec* x *une variable*
- *Termes :* $g(x) \longrightarrow f(a, f(b, x))$ *avec* x *une variable*

La réécriture

Pour définir *qui doit être réécrit en quoi*, nous parlons de règles de réécriture.

Exemple

- *Mots :* $ab \longrightarrow ba$ *ou* $ax \longrightarrow x$ *avec* x *une variable*
- *Termes :* $g(x) \longrightarrow f(a, f(b, x))$ *avec* x *une variable*

Un ensemble $\mathcal{R}$ *de règles de réécriture est appelé : Système de réécriture*

Réécriture de termes

Plus formellement,

Définition

Soient l et r deux termes de $\mathcal{T}(\mathcal{F}, \mathcal{X})$ tels que $\mathcal{V}ar(l) \subseteq \mathcal{V}ar(r)$, $l \to r$ est une règle de réécriture.

Start your engine!

La réécriture peut se faire à n'importe où dans le terme donné.

Start your engine!

La réécriture peut se faire à n'importe où dans le terme donné.

Exercice

Intuitivement, soit $f(f(a,x),h(x)) \rightarrow g(b,x)$ *une règle de réécriture, Que se passe-t'il si j'applique cette règle sur les termes suivants ?*

- *h(f(f(a,b),h(b)))*
- *f(f(a,b),h(b))*
- *h(f(f(a,a),h(b)))*
- *f(f(a,b),h(c))*

Application d'une règle de réécriture

Plus formellement...

Définition

Soient $l \to r$ une règle de réécriture et $t, t' \in \mathcal{T}(\mathcal{F})$ deux terme clos. t se réécrit en t' par $l \to r$, noté $t \overset{l \to r}{\to} t'$, s'il existe une position $\omega \in \mathcal{P}os(t)$ et une substitution $\sigma : \mathcal{X} \mapsto \mathcal{T}(\mathcal{F})$ telles que

- $t|_\omega = l\sigma$ *et*
- $t' = t[r\sigma]_\omega$.

Application d'une règle de réécriture

Plus formellement...

Définition

Soient $l \to r$ une règle de réécriture et $t, t' \in \mathcal{T}(\mathcal{F})$ deux terme clos. t se réécrit en t' par $l \to r$, noté $t \overset{l \to r}{\to} t'$, s'il existe une position $\omega \in \mathcal{P}os(t)$ et une substitution $\sigma : \mathcal{X} \mapsto \mathcal{T}(\mathcal{F})$ telles que

- $t|_\omega = l\sigma$ *et*
- $t' = t[r\sigma]_\omega$.

Pour un système de réécriture donné $\mathcal{R}$, on note $t \to_{\mathcal{R}} t'$ un pas de réécriture avec une des règles de $\mathcal{R}$.

Application d'une règle de réécriture

Plus formellement...

Définition

Soient $l \to r$ une règle de réécriture et $t, t' \in \mathcal{T}(\mathcal{F})$ deux terme clos. t se réécrit en t' par $l \to r$, noté $t \overset{l \to r}{\to} t'$, s'il existe une position $\omega \in \mathcal{P}os(t)$ et une substitution $\sigma : \mathcal{X} \mapsto \mathcal{T}(\mathcal{F})$ telles que

- $t|_\omega = l\sigma$ *et*
- $t' = t[r\sigma]_\omega$.

Pour un système de réécriture donné $\mathcal{R}$, on note $t \to_{\mathcal{R}} t'$ un pas de réécriture avec une des règles de $\mathcal{R}$.

$t \to^*_{\mathcal{R}} t'$ décrit le fait que t' est atteignable par réécriture à partir de t avec un nombre fini d'étapes.

Quelques exemples

Exercice

- *Montrer que d est atteignable par réécriture à partir de* $f(b, g(c))$ *avec le système de réécriture* $\mathcal{R} = \{f(x, g(d)) \to f(c, g(d)), g(x) \to x, c \to d, f(x, x) \to x\}$.
- *Appliquer* $\mathcal{R} = \{add(succ(x), y) \to add(x, succ(y)), add(zero, x) \to x\}$ *sur le terme* $add(succ(succ(0)), succ(succ(0)))$ *tant que possible. Que représentent finalement ces deux règles ?*
- *Soient les constructeurs de listes cons et nil. Définir un système de réécriture dont le but est de supprimer toutes les occurences de la constante a dans une liste quelconque.*

Ensemble des accessibles par réécriture

L'ensemble des termes atteignables par réécriture à partir d'un ensemble de termes E et avec le système de réécriture $\mathcal{R}$ est noté $\mathcal{R}^*(E)$

Ensemble des accessibles par réécriture

L'ensemble des termes atteignables par réécriture à partir d'un ensemble de termes E et avec le système de réécriture $\mathcal{R}$ est noté $\mathcal{R}^*(E)$
Plus formellement,

$$\mathcal{R}^*(E) = \{t \in \mathcal{T}(\mathcal{F}) \mid \exists t_0 \in E.t_0 \rightarrow_{\mathcal{R}}^* t\}$$

La question cruciale en réécriture

$$t \rightarrow_{\mathcal{R}}^{*} t' ?$$

La question cruciale en réécriture

$$t \rightarrow_{\mathcal{R}}^{*} t' ?$$

Réponse : Le problème d'atteignabilité en réécriture est indécidable

Un petit exemple problématique

Pour $\mathcal{R} = \{g(x, y) \to g(s(x), s(y))\}$ et $t = g(0, 0)$, $\mathcal{R}^*(\{t\})$ n'est pas un langage régulier.

Si c'est indécidable, alors il n'y a rien à faire ?

Si c'est indécidable, alors il n'y a rien à faire ?

Au contraire !

Si c'est indécidable, alors il n'y a rien à faire ?

Au contraire !

- Trouver les classes de systèmes de réécriture pour lesquelles ce problème est décidable
  - $\mathcal{R}$ est clos (Dauchet, Tison 1990, Brainerd 1969)
  - $\mathcal{R}$ est linéaire à droite et monadique (Salomaa, 1988)
  - $\mathcal{R}$ est linéaire et semi-monadique (Coquidé et al., 1991)
  - ...

Si c'est indécidable, alors il n'y a rien à faire ?

Au contraire !

- Trouver les classes de systèmes de réécriture pour lesquelles ce problème est décidable
  - $\mathcal{R}$ est clos (Dauchet, Tison 1990, Brainerd 1969)
  - $\mathcal{R}$ est linéaire à droite et monadique (Salomaa, 1988)
  - $\mathcal{R}$ est linéaire et semi-monadique (Coquidé et al., 1991)
  - ...
- Le problème de terminaison d'un système de réécriture

Si c'est indécidable, alors il n'y a rien à faire ?

Au contraire !

- Trouver les classes de systèmes de réécriture pour lesquelles ce problème est décidable
  - $\mathcal{R}$ est clos (Dauchet, Tison 1990, Brainerd 1969)
  - $\mathcal{R}$ est linéaire à droite et monadique (Salomaa, 1988)
  - $\mathcal{R}$ est linéaire et semi-monadique (Coquidé et al., 1991)
  - ...
- Le problème de terminaison d'un système de réécriture
- Définir des semi-algorithmes efficaces pour explorer l'ensemble des accessibles

Si c'est indécidable, alors il n'y a rien à faire ?

Au contraire !

- Trouver les classes de systèmes de réécriture pour lesquelles ce problème est décidable
  - $\mathcal{R}$ est clos (Dauchet, Tison 1990, Brainerd 1969)
  - $\mathcal{R}$ est linéaire à droite et monadique (Salomaa, 1988)
  - $\mathcal{R}$ est linéaire et semi-monadique (Coquidé et al., 1991)
  - ...
- Le problème de terminaison d'un système de réécriture
- Définir des semi-algorithmes efficaces pour explorer l'ensemble des accessibles
- Définir des techniques d'approximation pour prouver qu'un terme (ou ensemble de termes) n'est pas atteignable

Et on a pas encore parlé du pouvoir d'expression

Et on a pas encore parlé du pouvoir d'expression

Les systèmes de réécriture sont Turing-complets

Et on a pas encore parlé du pouvoir d'expression

Les systèmes de réécriture sont Turing-complets

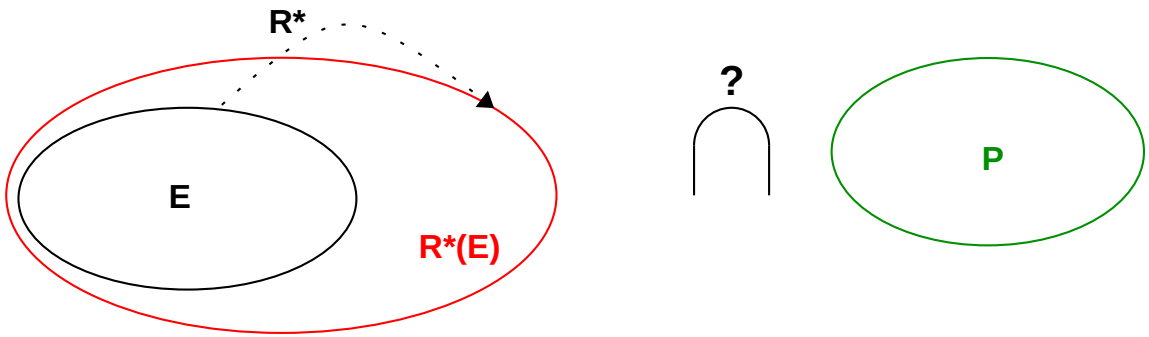
- Les règles de réécriture = action/réaction (ou réception/envoi ou avant/après)

Et on a pas encore parlé du pouvoir d'expression

Les systèmes de réécriture sont Turing-complets

- Les règles de réécriture = action/réaction (ou réception/envoi ou avant/après)
- Spécification d'algorithmes en réécriture : sémantique exécutable

Vérification et Réécriture

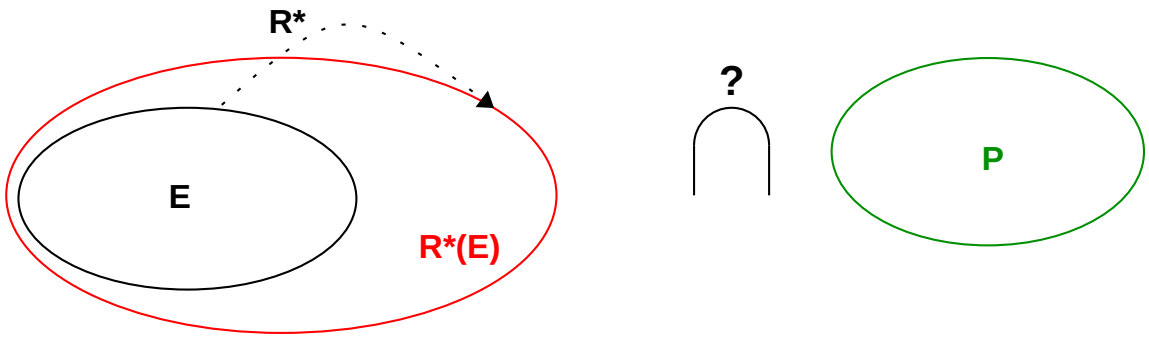


E : termes initiaux

R : système de réécriture

P : termes cibles

Vérification et Réécriture



E : termes initiaux ⟶ **Configurations initiales du système**

R : système de réécriture ⟶ **Evolution du système**

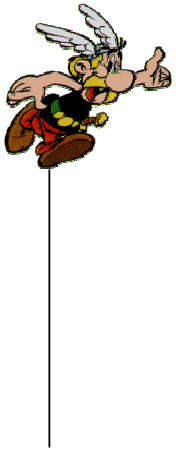
P : termes cibles ⟶ **Configurations interdites**

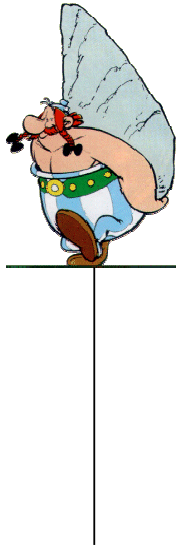
Outline

Qu'est ce qu'un protocole de sécurité ?

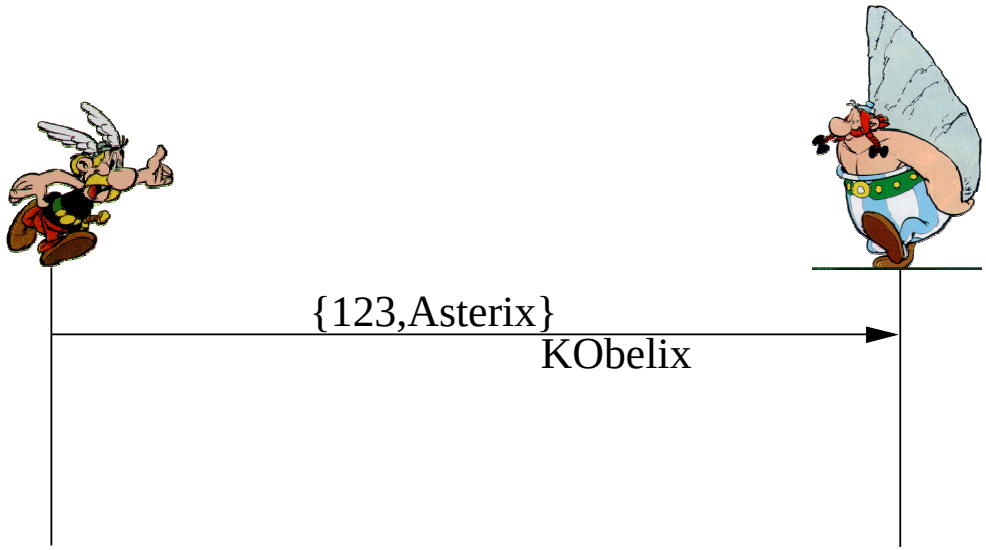
- But : Echange de messages afin de sécuriser des communications entre individus (ex : phase précédant un paiement en ligne).
- Propriétés à vérifier :
  - Confidentialité
  - Authentification
  - Non-répudiation
  - ...
- Problème de sécurité des protocoles indécidable en général
- Problème de sécurité peut se réduire en un problème d'atteignabilité en réécriture.

Needham-Schroeder

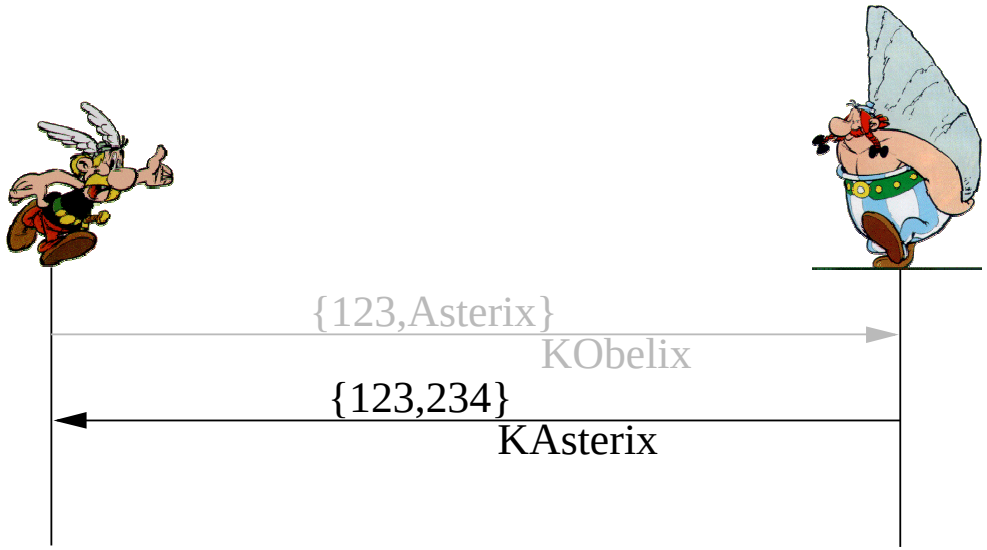




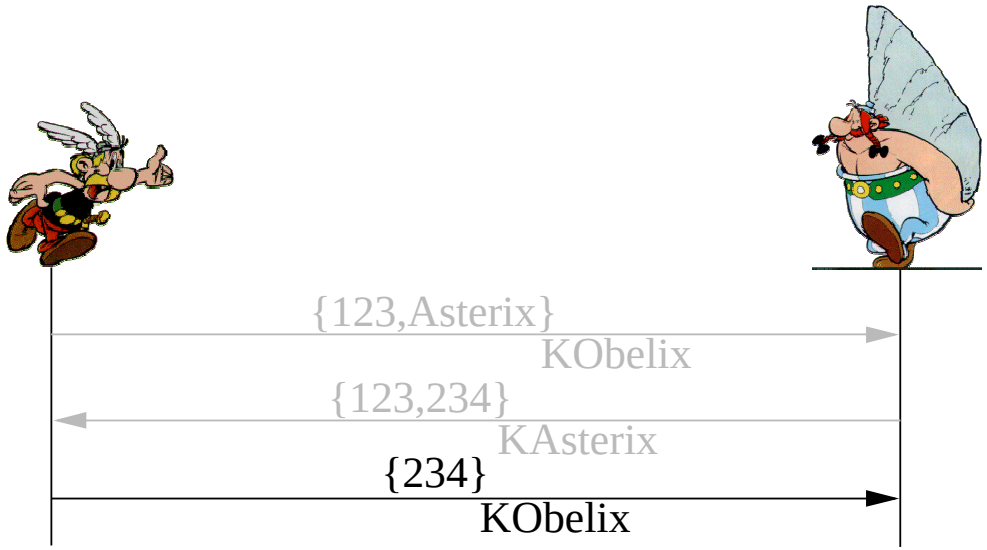
Needham-Schroeder



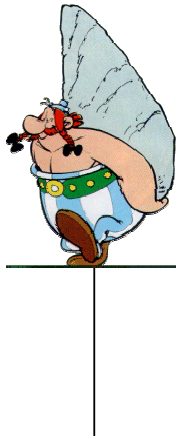
Needham-Schroeder



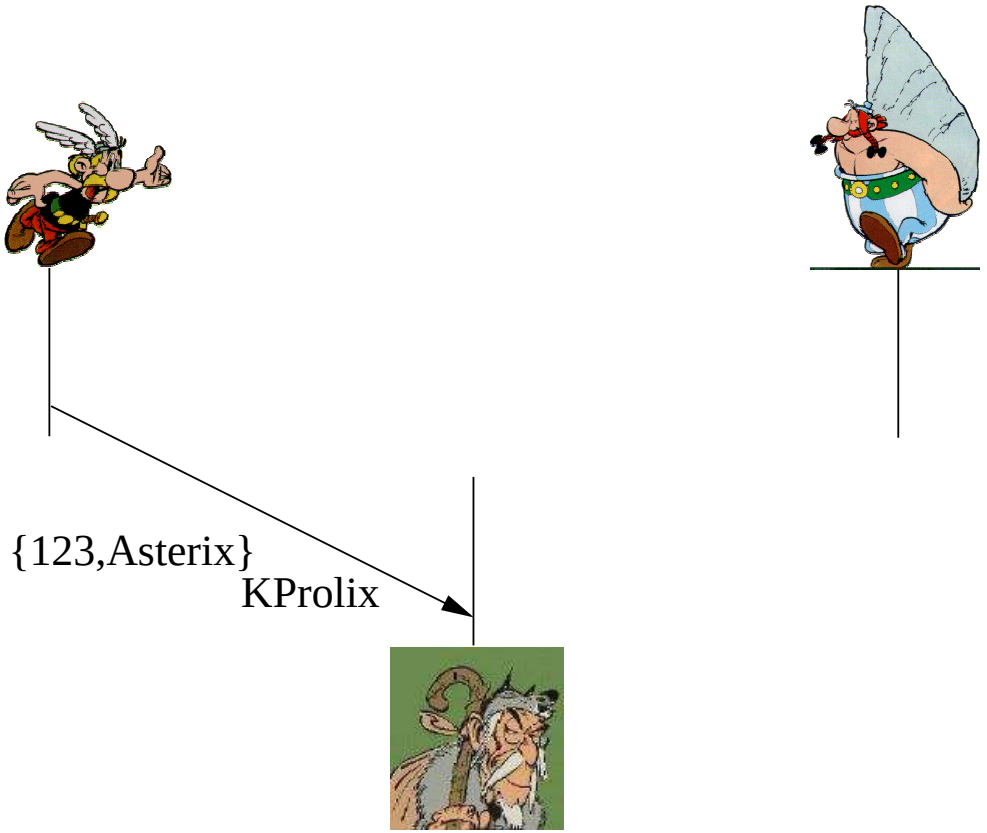
Needham-Schroeder



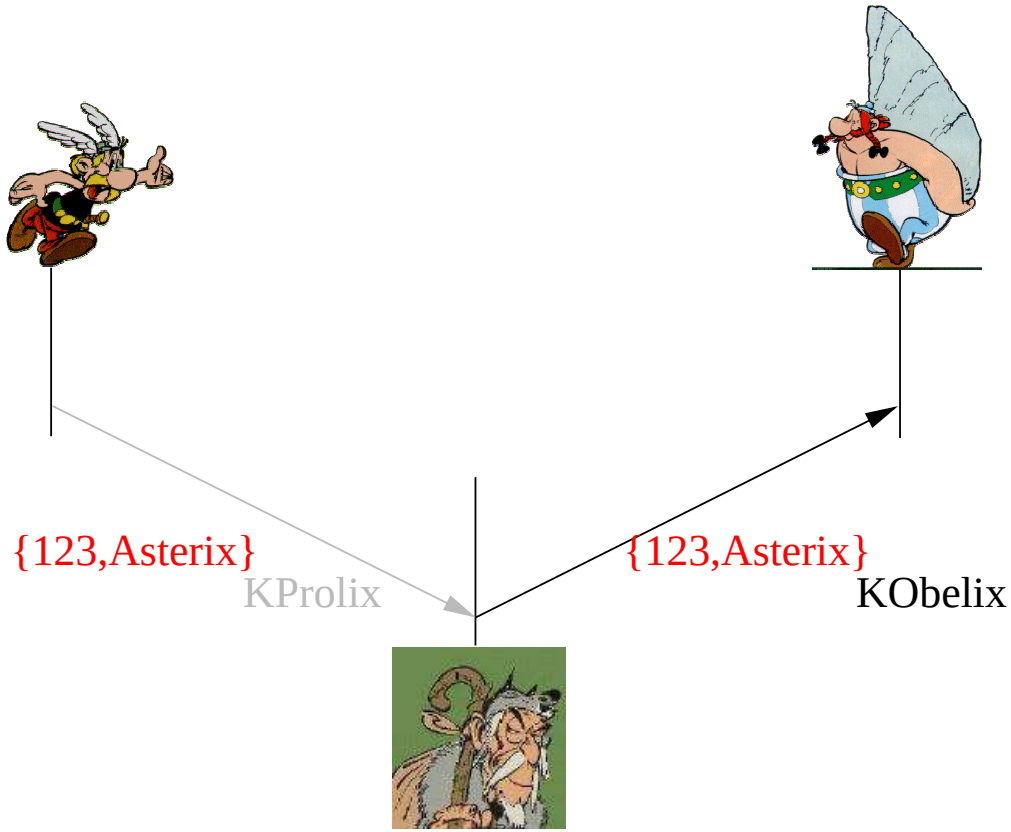
Attaque contre Needham-Schroeder



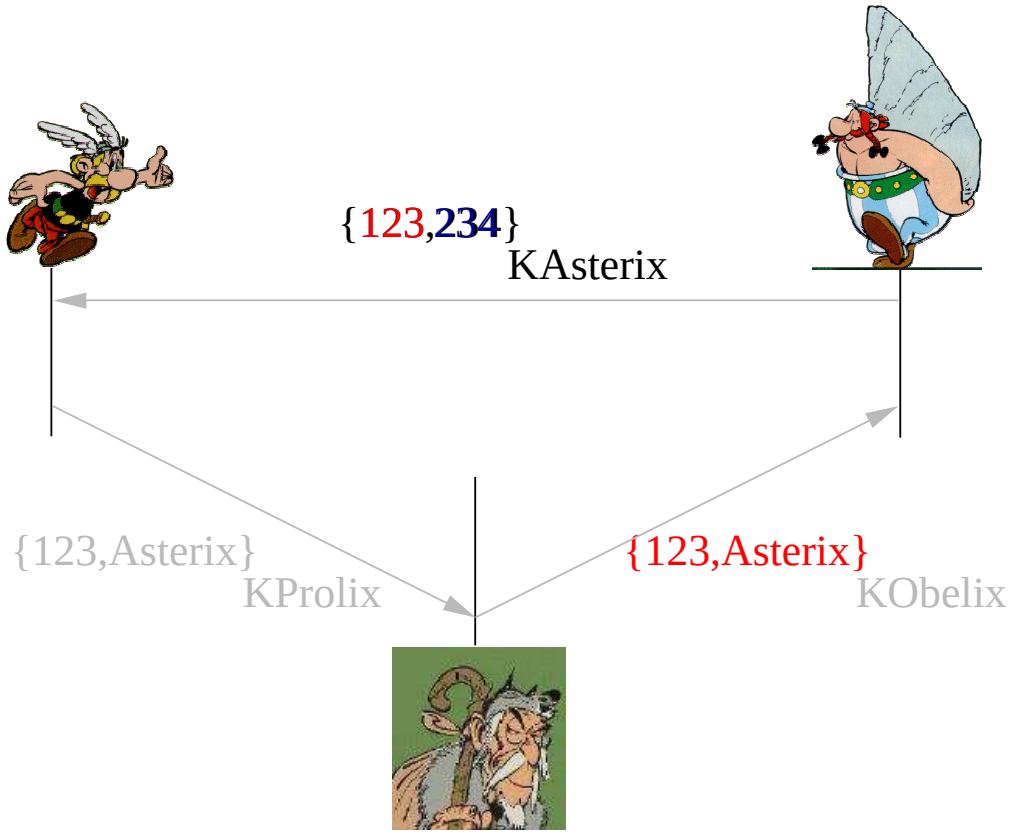
Attaque contre Needham-Schroeder



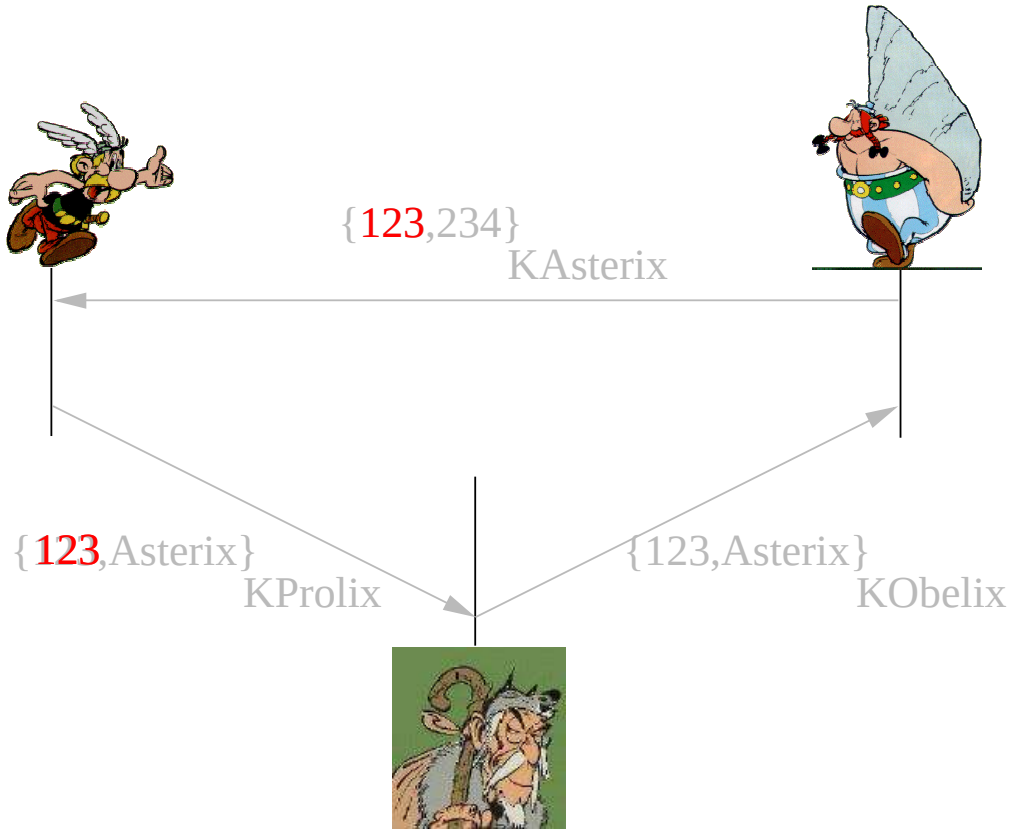
Attaque contre Needham-Schroeder



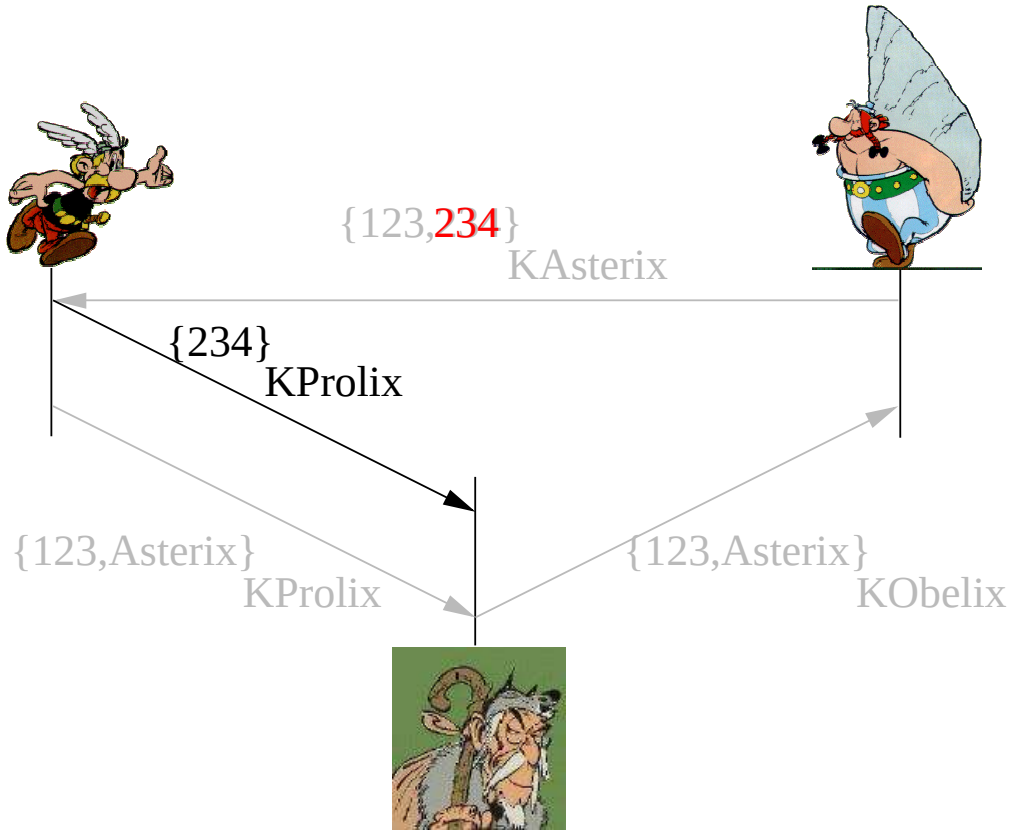
Attaque contre Needham-Schroeder



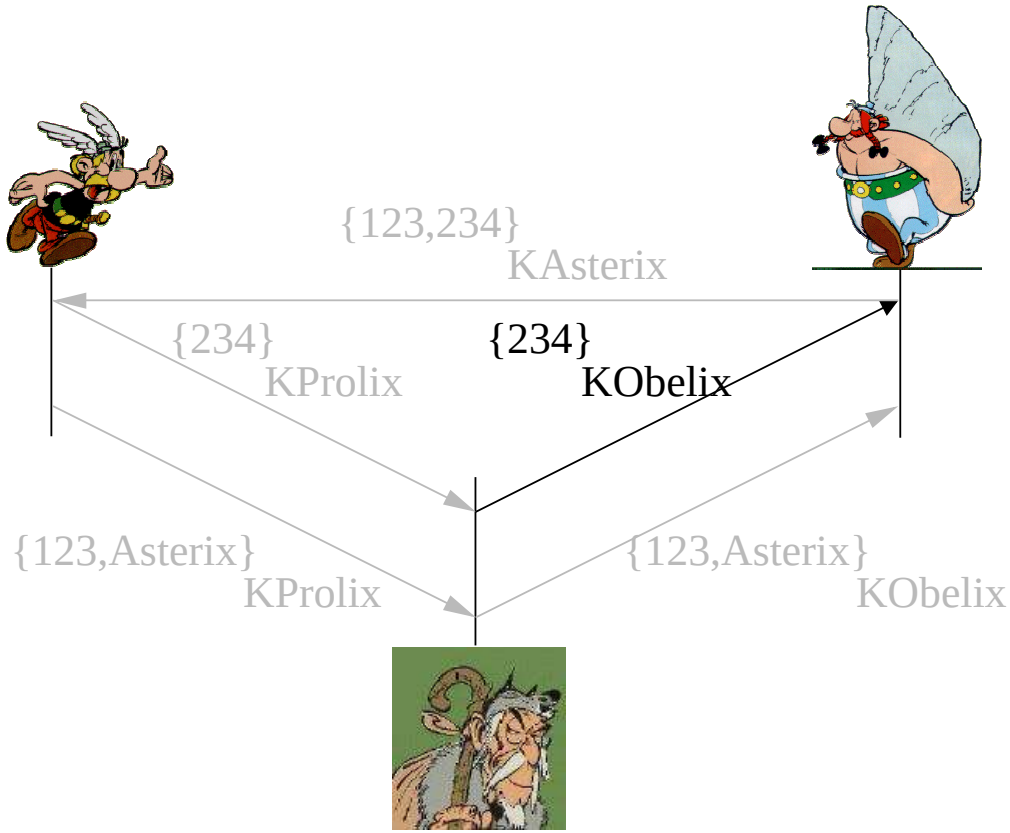
Attaque contre Needham-Schroeder



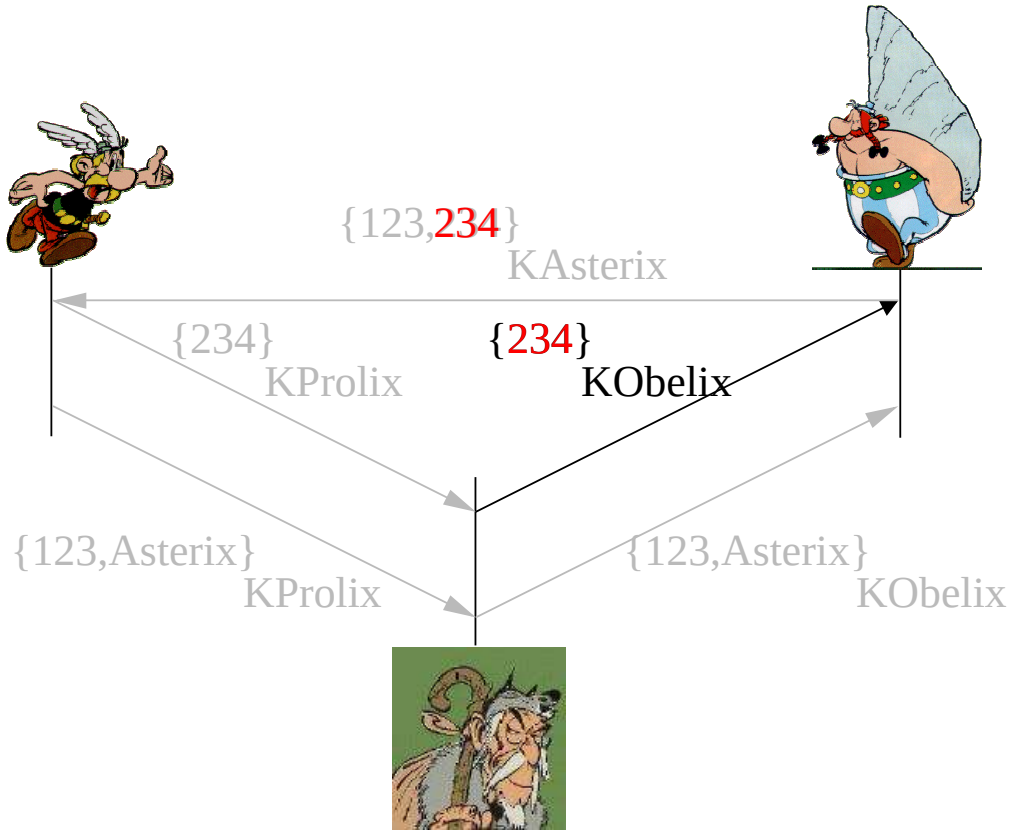
Attaque contre Needham-Schroeder



Attaque contre Needham-Schroeder



Attaque contre Needham-Schroeder



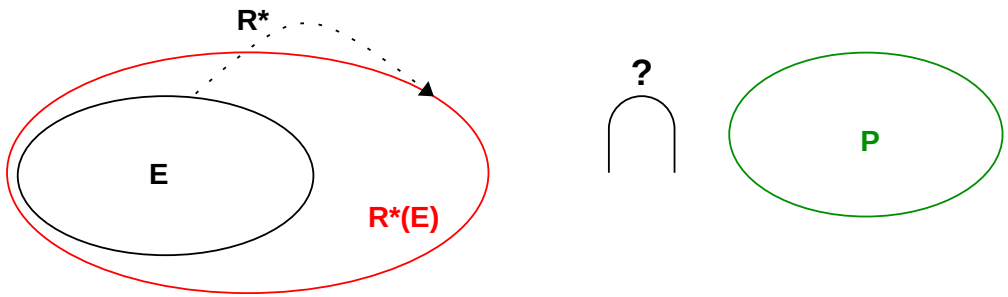
Modélisation pour la vérification

- Système de réécriture $\mathcal{R}$:
  - Différentes étapes du protocole

$$goal(As, Ob) \longrightarrow msg(As, Ob, crypt(pk(Ob), pair(n(As, Ob), id(As))))$$

  - Pouvoir de l'intrus
    1. Décodage : $U(inv(pk(x)), crypt(pk(x), y)) \longrightarrow y$;
    2. Décomposition : $pair(x, y) \longrightarrow y$;
    3. Composition : $U(x, y) \longrightarrow pair(x, y)$;
    4. ...
  - Tout message envoyé est connu par l'intrus : $msg(x, y, z) \longrightarrow z$.
- Ensemble de terme E :
  - E = Connaissance de l'intrus = Configuration du réseau.

Vérification de propriétés de secret



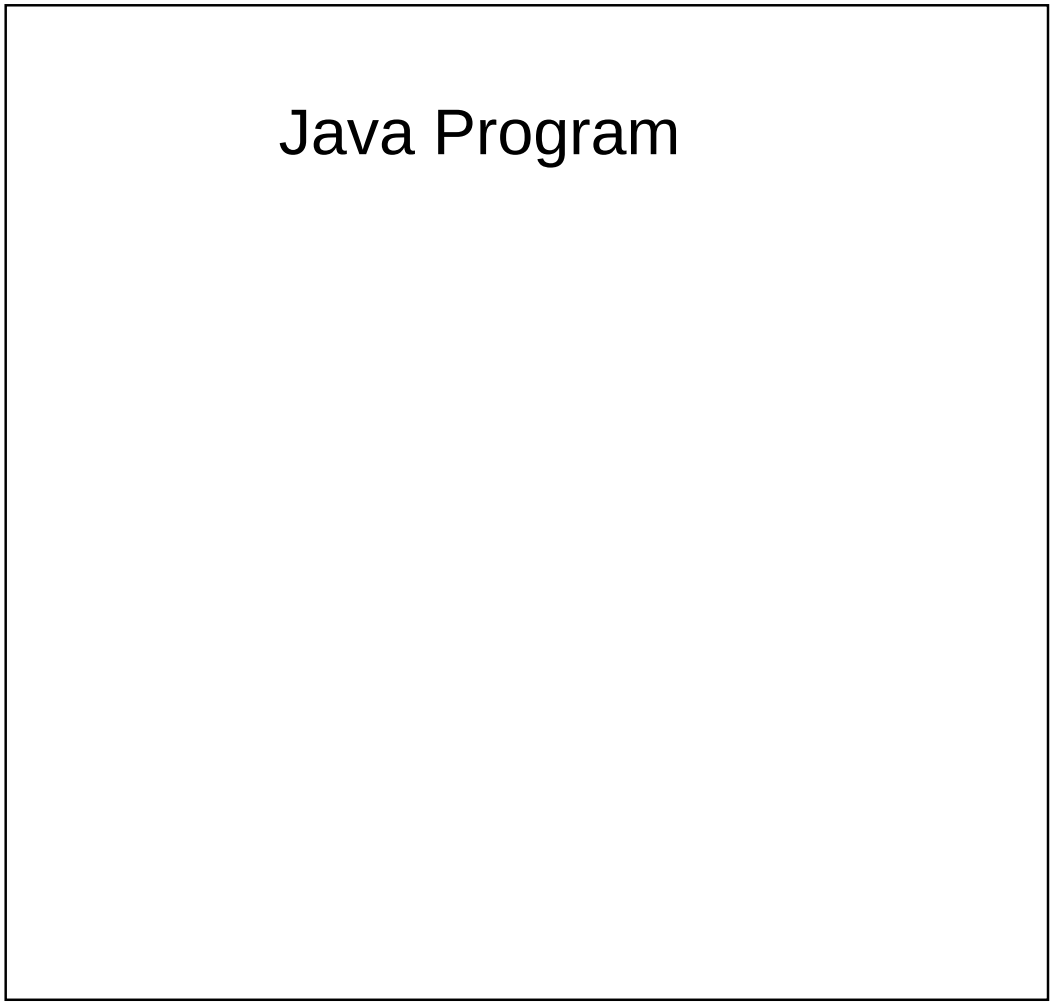
E : termes initiaux → **Connaissance initiale de l'intrus**

R : système de réécriture → **Protocole + pouvoir de l'intrus**

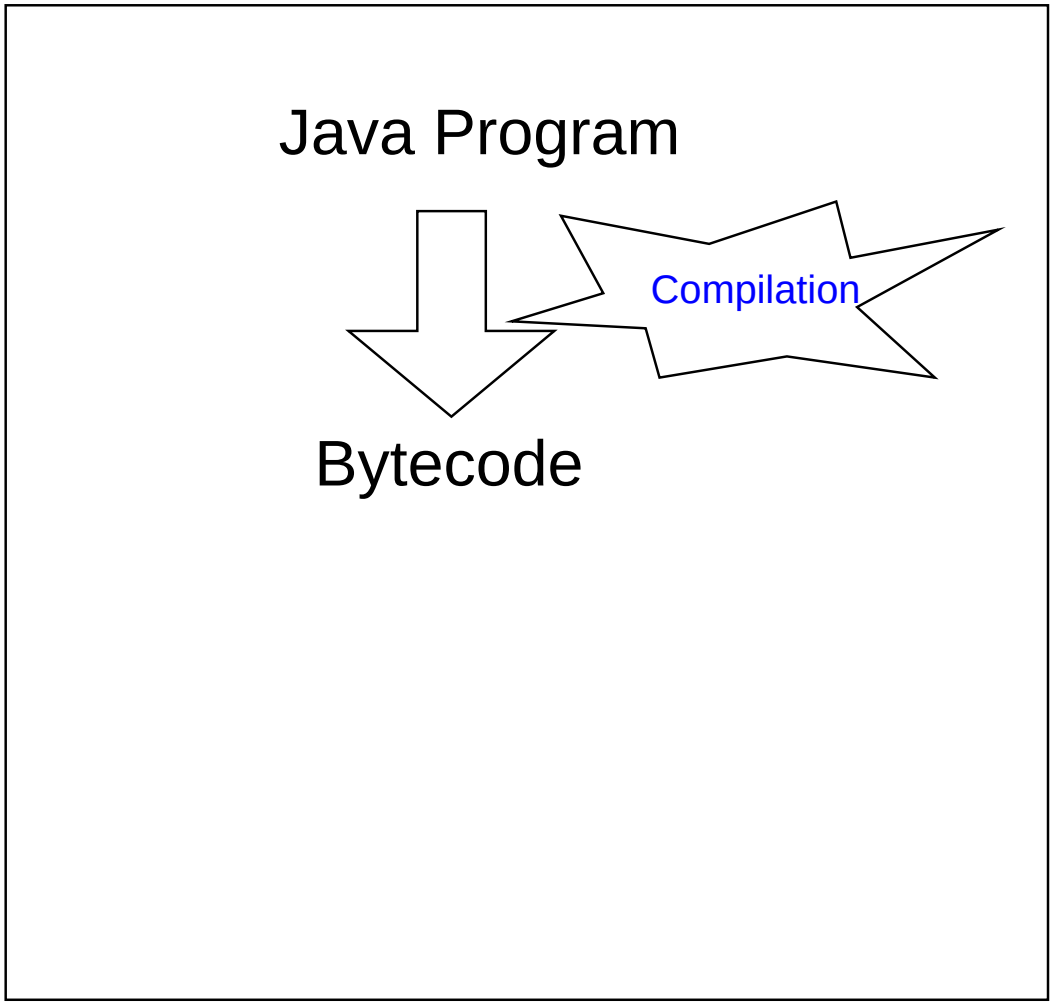
P : termes cibles → **Termes secrets**

Outline

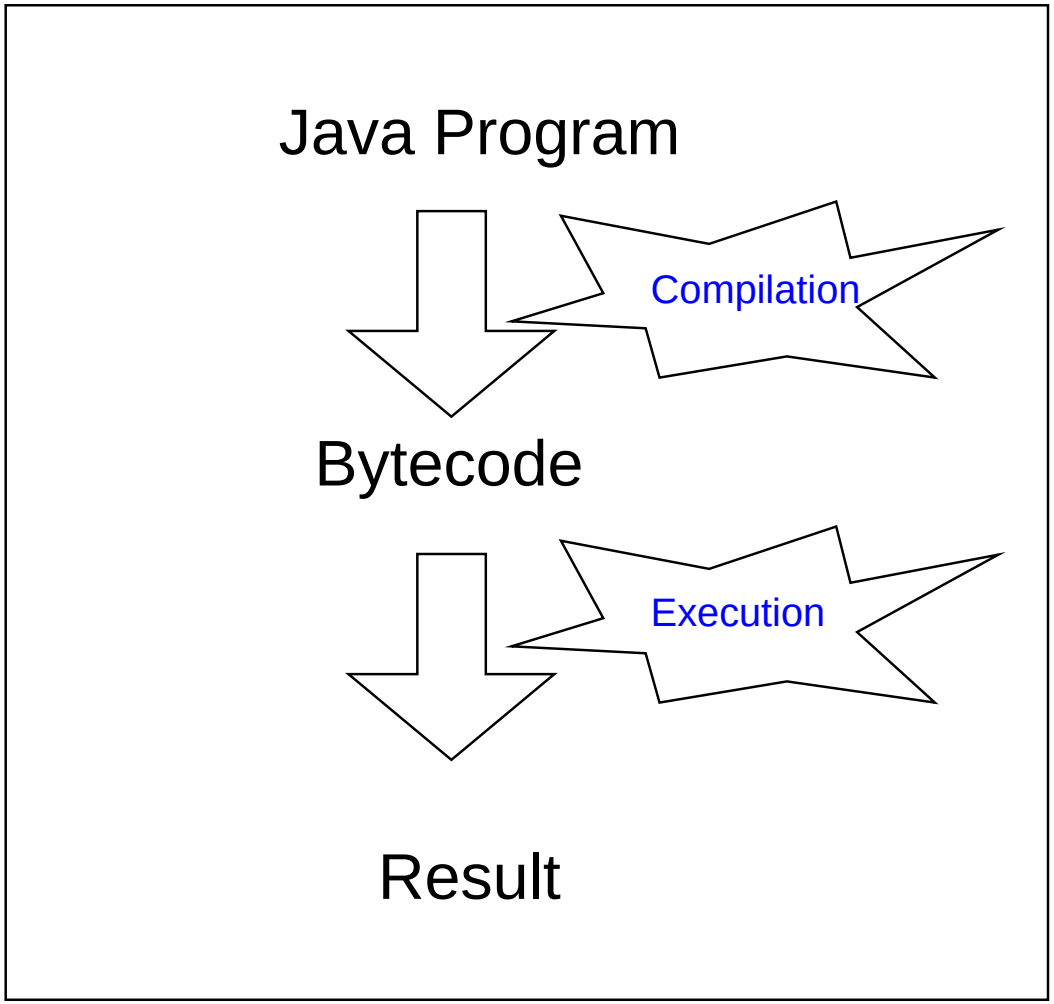
Contexte



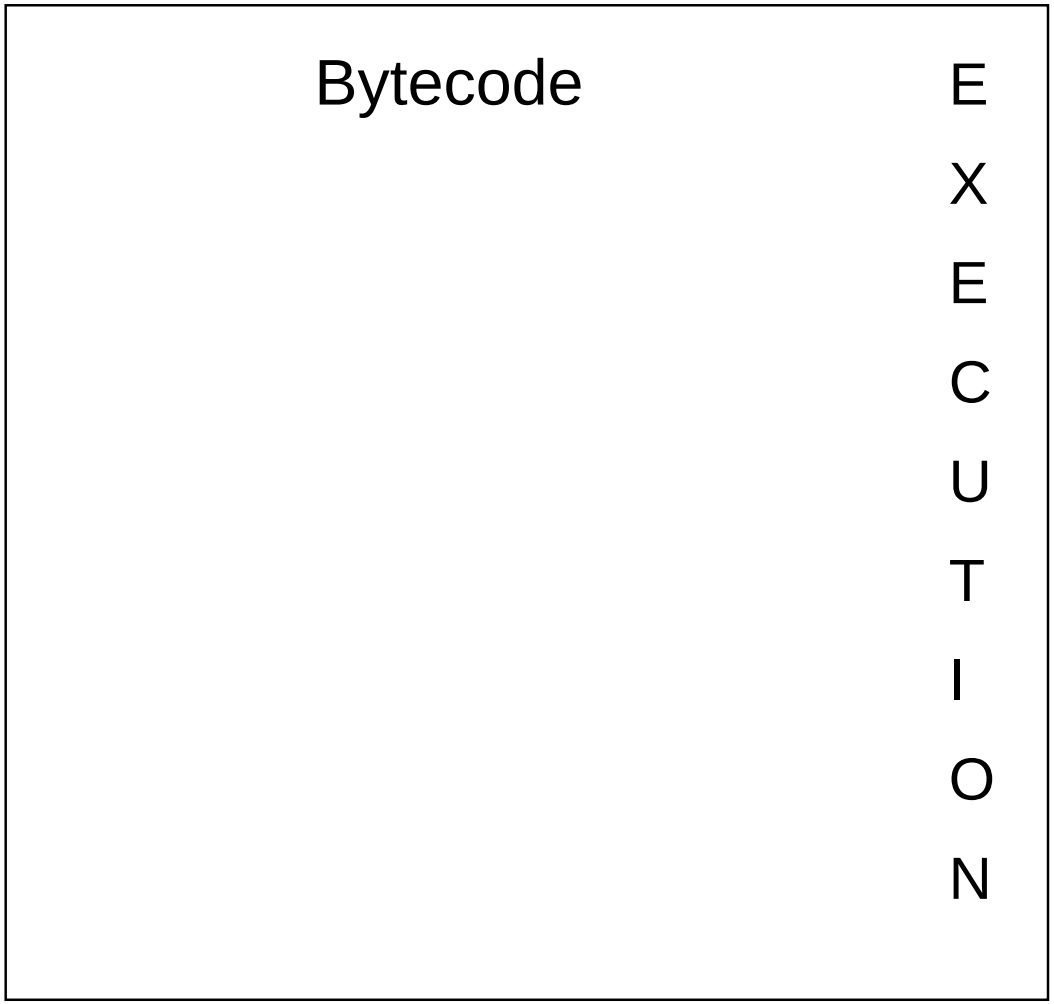
Contexte



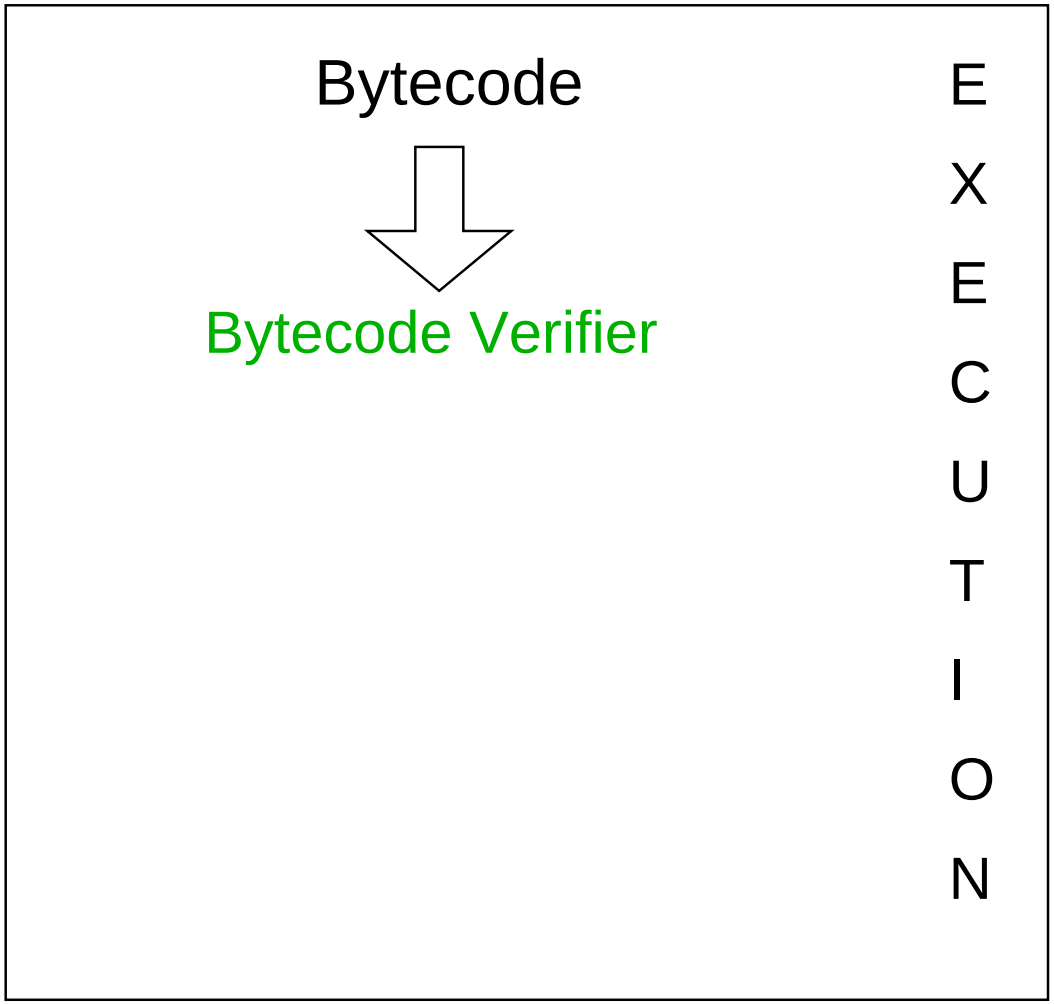
Contexte



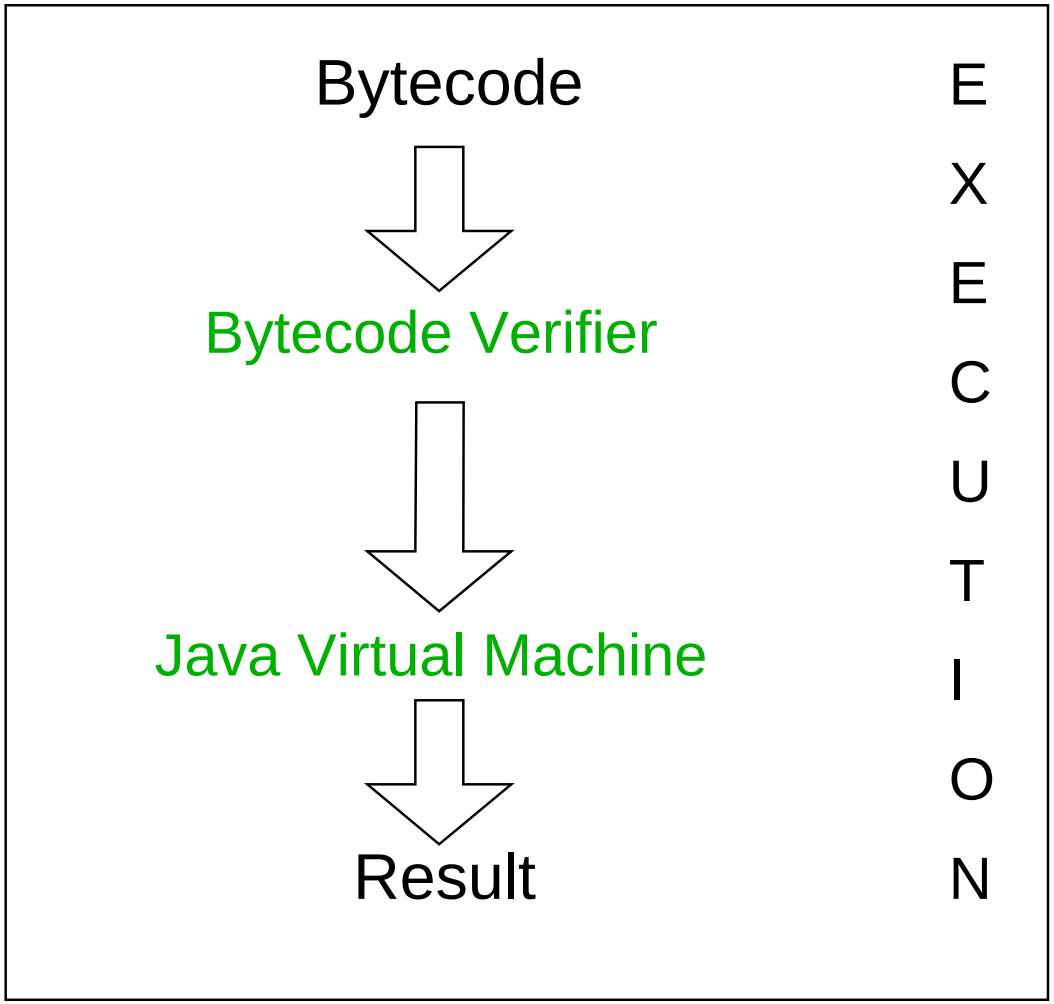
Contexte



Contexte



Contexte


Bytecode
Bytecode Verifier
Java Virtual Machine
Result
EXECUTION

Bytecode Verifier assure . . .

- Pas de débordements de pile
- Types des paramètres des instructions bytecode corrects
- Accès aux champs des objets légaux – privé, publique, ou protégé

Bytecode Verifier assure . . .

- Pas de débordements de pile
- Types des paramètres des instructions bytecode corrects
- Accès aux champs des objets légaux – privé, publique, ou protégé

Vérification de propriétés de sûreté

Ce que nous avons fait...

Objectifs

- Sémantique de réécriture pour le bytecode Java
- Analyse statique à partir de l'analyse d'atteignabilité en réécriture

Ce que nous avons fait...

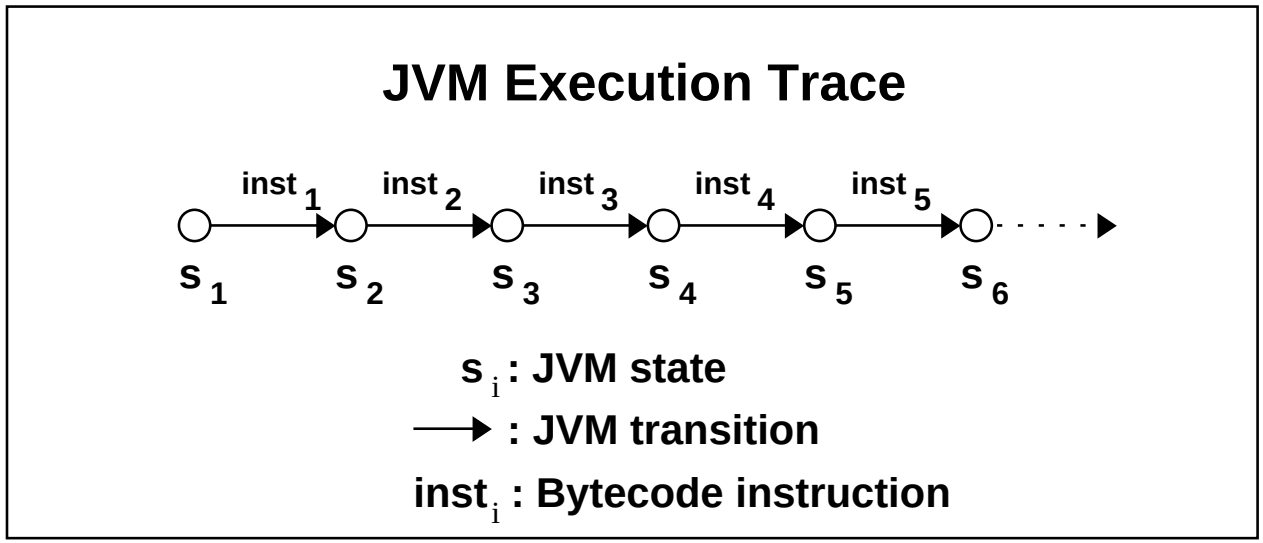
Objectifs

- Sémantique de réécriture pour le bytecode Java
- Analyse statique à partir de l'analyse d'atteignabilité en réécriture

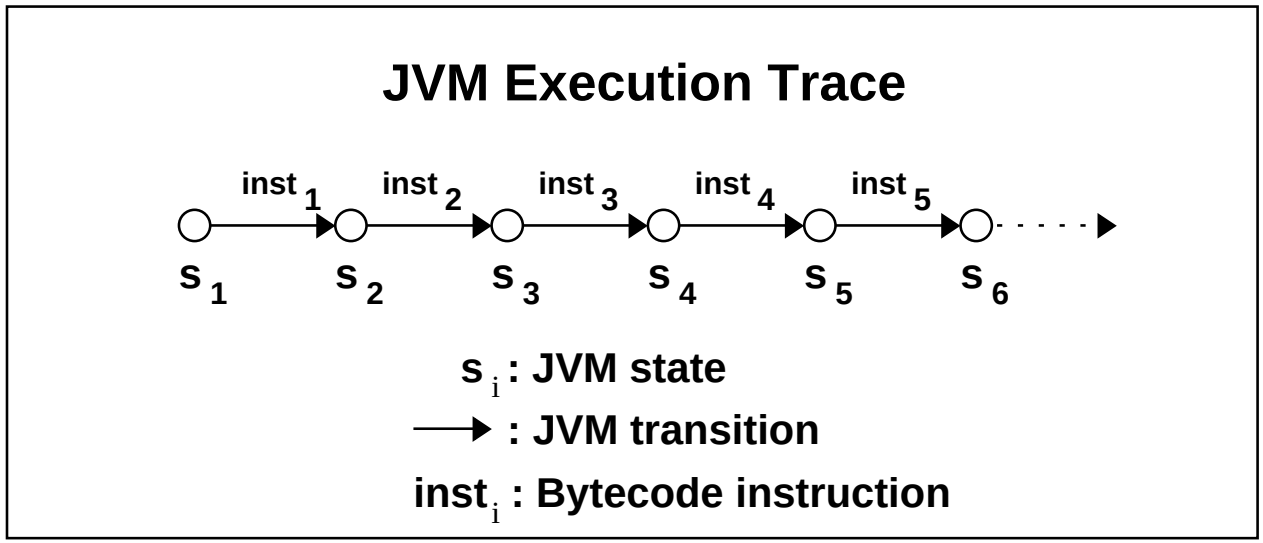
Notre proposition

- Etat de la JVM - Termes
- Instructions bytecode - règles de réécriture
- Vérification de propriétés de sûreté = problème d'atteignabilité

Quelques mots sur la JVM

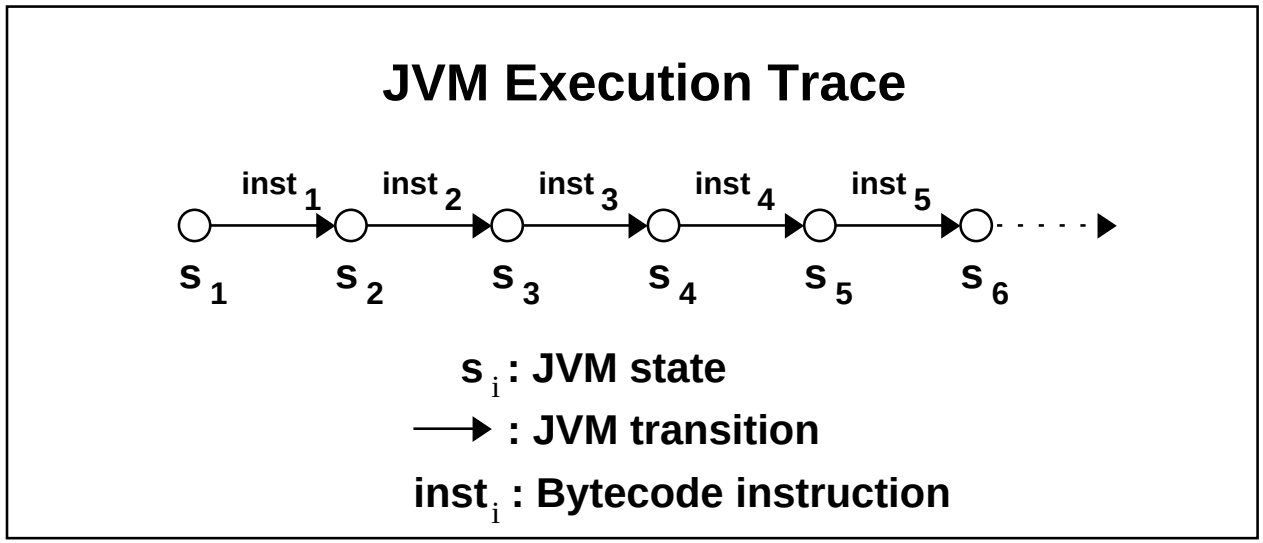


Quelques mots sur la JVM



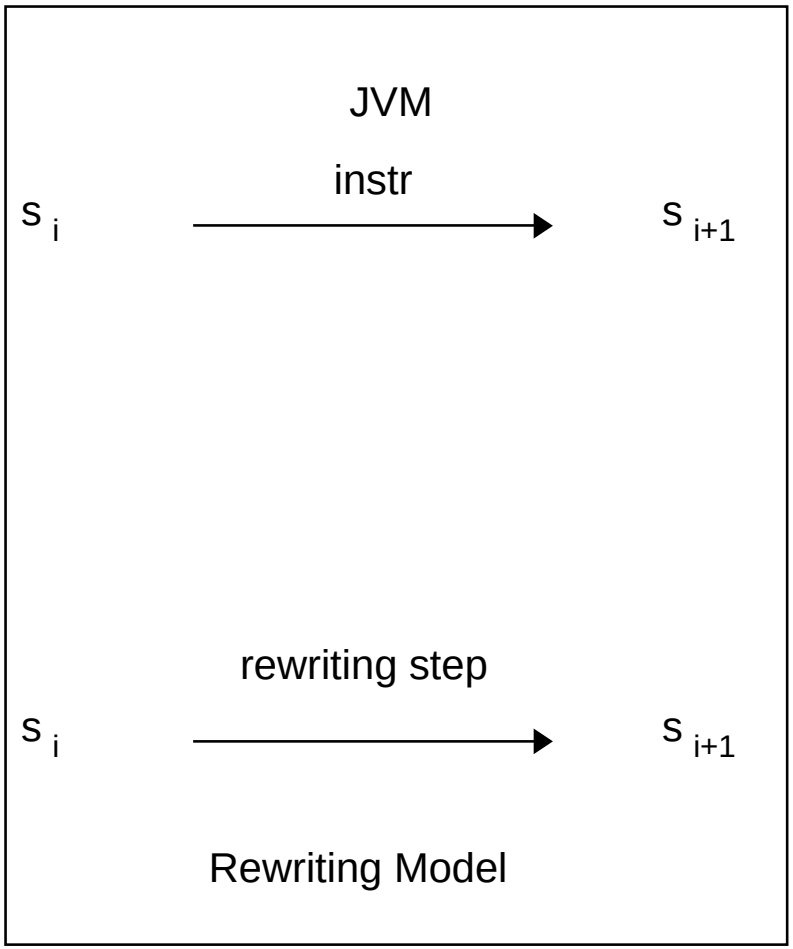
- Frame : $\langle$name, program point, operand stack, local variables$\rangle$

Quelques mots sur la JVM



- Frame : ⟨name, program point, operand stack, local variables⟩
- Etat JVM : ⟨current frame, frame stack, heap, static heap ⟩
  - Tas (heap) aloue une adresse à un objet
  - Tas statique - valeurs statiques

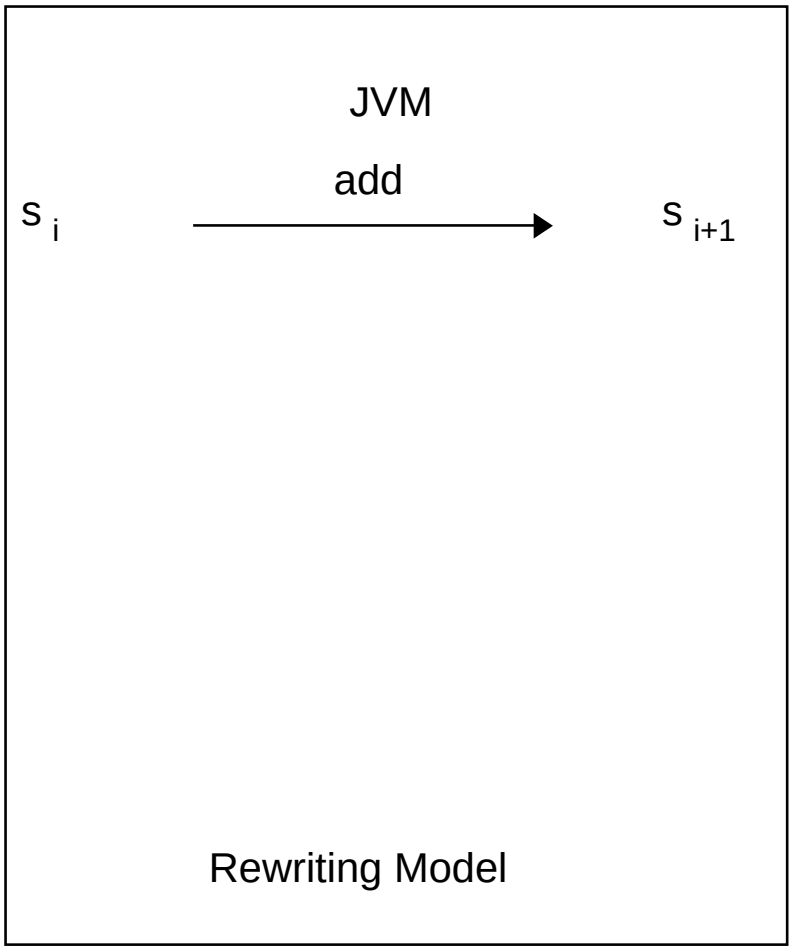
Transitions JVM – Règles de réécriture



En général, une transition JVM de s_i à s_{i+1} ne correspond pas à un simple pas de réécriture

Pourquoi ?

Transitions JVM – Règles de réécriture

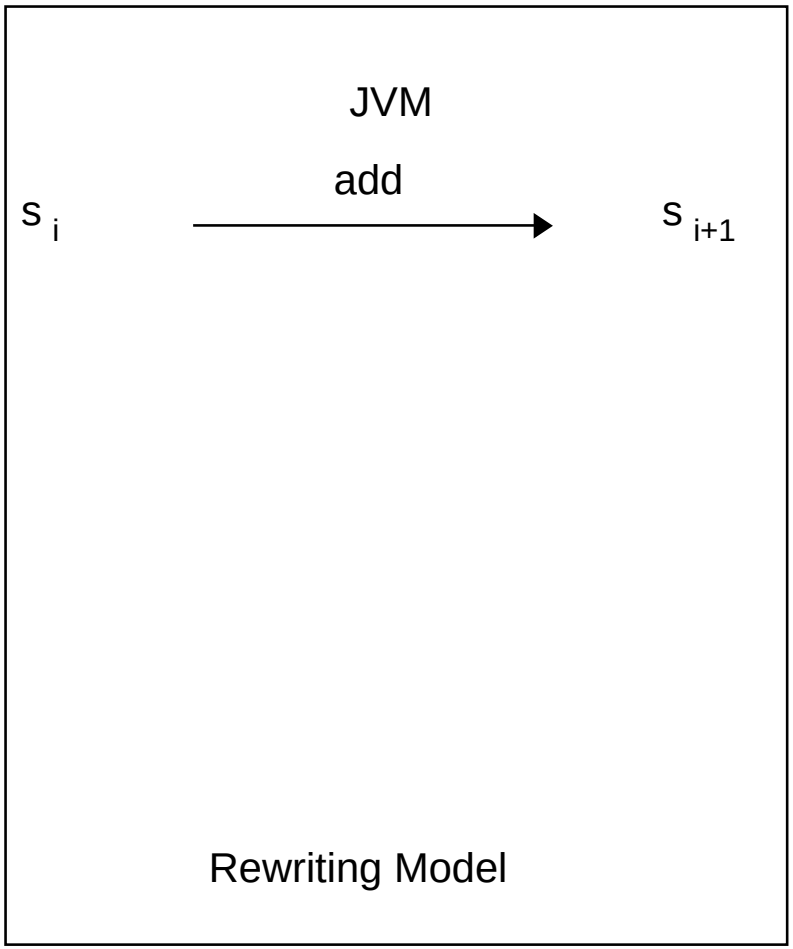


En général, une transition JVM de s_i à s_{i+1} ne correspond pas à un simple pas de réécriture

Pourquoi ?

add: $\dfrac{s_i = \langle\langle m, pc, x::y::os, l\rangle, fs, h, sh\rangle}{s_{i+1} = \langle\langle m, pc+1, (x+y)::os, l\rangle, fs, h, sh\rangle}$

Transitions JVM – Règles de réécriture



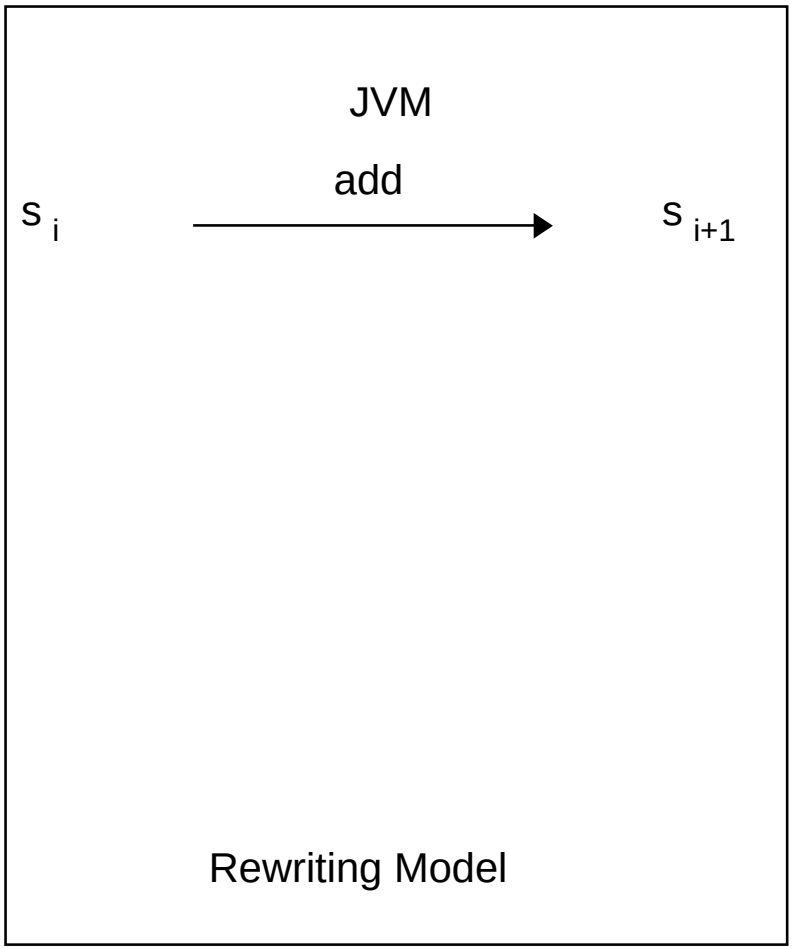
En général, une transition JVM de s_i à s_{i+1} ne correspond pas à un simple pas de réécriture

Pourquoi ?

add: $\frac{s_i = \langle\langle m, pc, x::y::os, l\rangle, fs, h, sh\rangle}{s_{i+1} = \langle\langle m, pc+1, (x+y)::os, l\rangle, fs, h, sh\rangle}$

En réécriture, $(x+y)$ doit être évalué avant d'être stocké dans la pile os.

Transitions JVM – Règles de réécriture



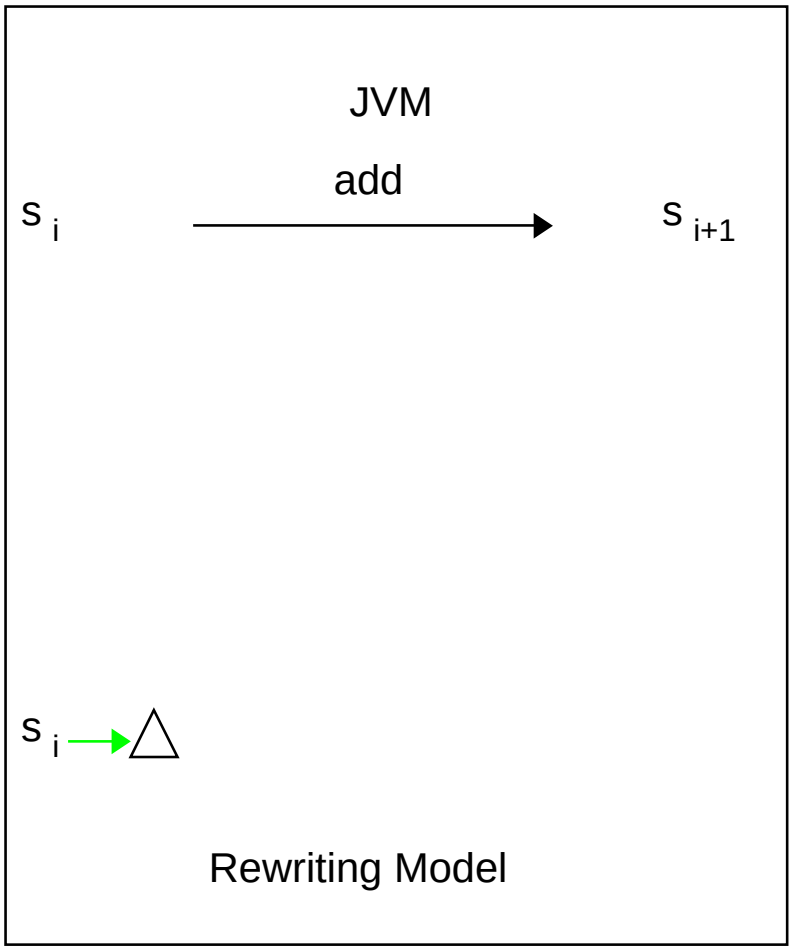
$frame(name(foo, A), pp2, s, l) \rightarrow$
$xframe(add, name(foo, A), pp2, s, l)$

$xframe(add, m, pc, stack(b, stack(a, s)), l) \rightarrow$
$xframe(xadd(a, b), m, pc, s, l)$

$xadd(succ(a), b) \rightarrow xadd(a, succ(b))$
$xadd(zero, b) \rightarrow result(b)$

$xframe(result(x), m, pc, s, l) \rightarrow$
$frame(m, next(pc), stack(x, s), l)$

Transitions JVM – Règles de réécriture



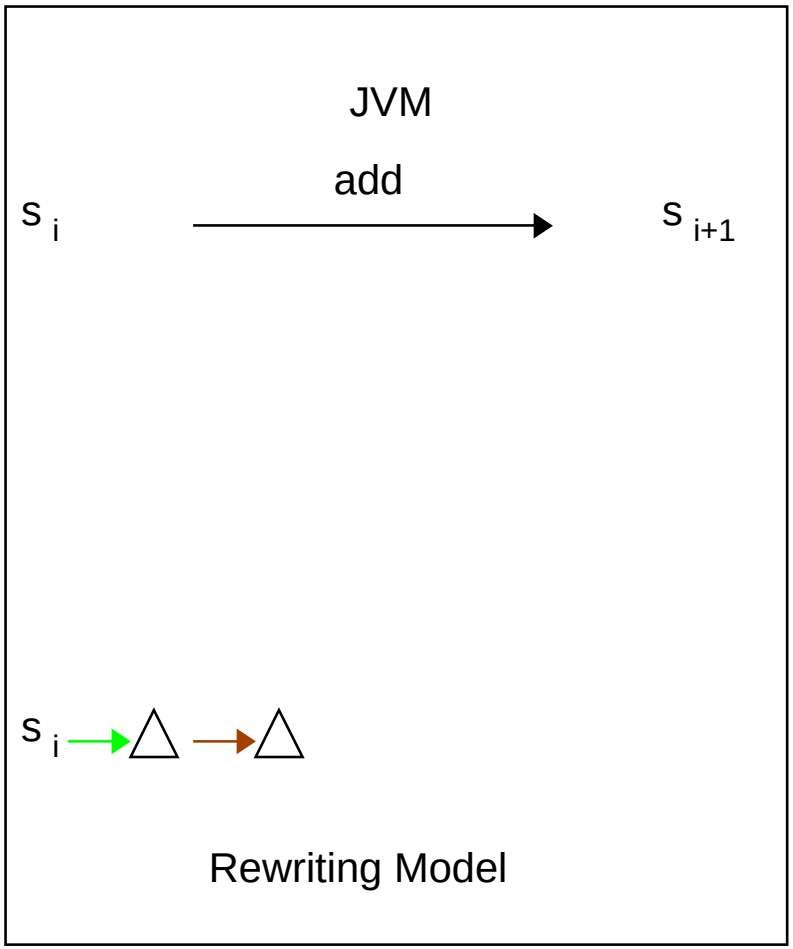
$frame(name(foo, A), pp2, s, l) \rightarrow$
$xframe(add, name(foo, A), pp2, s, l)$

$xframe(add, m, pc, stack(b, stack(a, s)), l) \rightarrow$
$xframe(xadd(a, b), m, pc, s, l)$

$xadd(succ(a), b) \rightarrow xadd(a, succ(b))$
$xadd(zero, b) \rightarrow result(b)$

$xframe(result(x), m, pc, s, l) \rightarrow$
$frame(m, next(pc), stack(x, s), l)$

Transitions JVM – Règles de réécriture



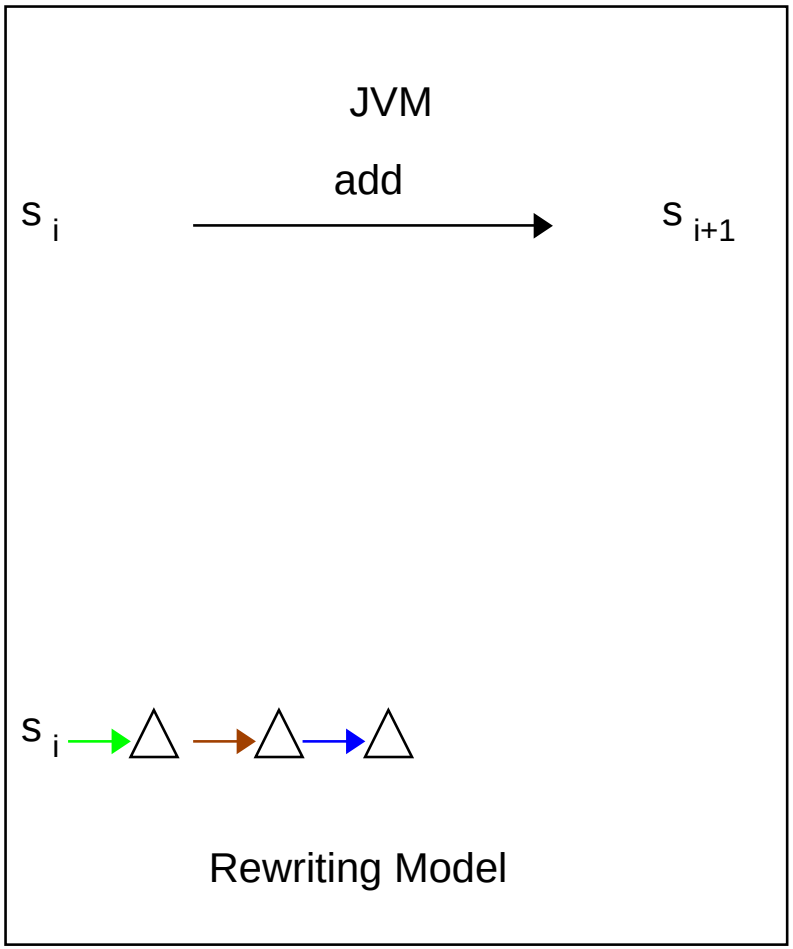
$frame(name(foo, A), pp2, s, l) \rightarrow$
$xframe(add, name(foo, A), pp2, s, l)$

$xframe(add, m, pc, stack(b, stack(a, s)), l) \rightarrow$
$xframe(xadd(a, b), m, pc, s, l)$

$xadd(succ(a), b) \rightarrow xadd(a, succ(b))$
$xadd(zero, b) \rightarrow result(b)$

$xframe(result(x), m, pc, s, l) \rightarrow$
$frame(m, next(pc), stack(x, s), l)$

Transitions JVM – Règles de réécriture



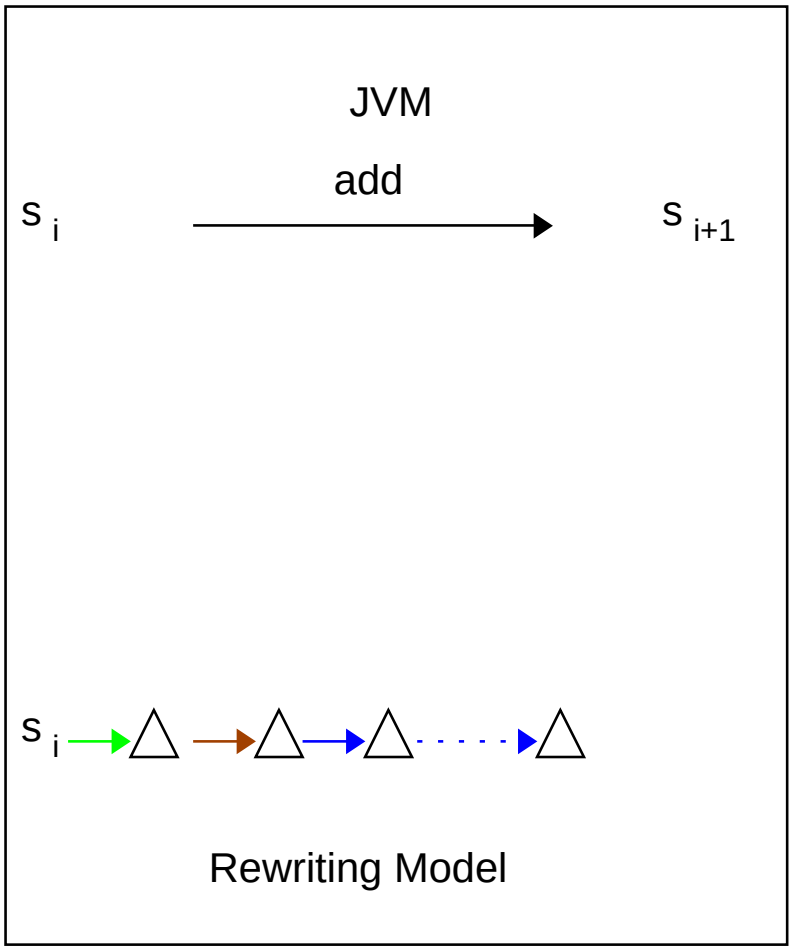
$frame(name(foo, A), pp2, s, l) \rightarrow$
$xframe(add, name(foo, A), pp2, s, l)$

$xframe(add, m, pc, stack(b, stack(a, s)), l) \rightarrow$
$xframe(xadd(a, b), m, pc, s, l)$

$xadd(succ(a), b) \rightarrow xadd(a, succ(b))$
$xadd(zero, b) \rightarrow result(b)$

$xframe(result(x), m, pc, s, l) \rightarrow$
$frame(m, next(pc), stack(x, s), l)$

Transitions JVM – Règles de réécriture



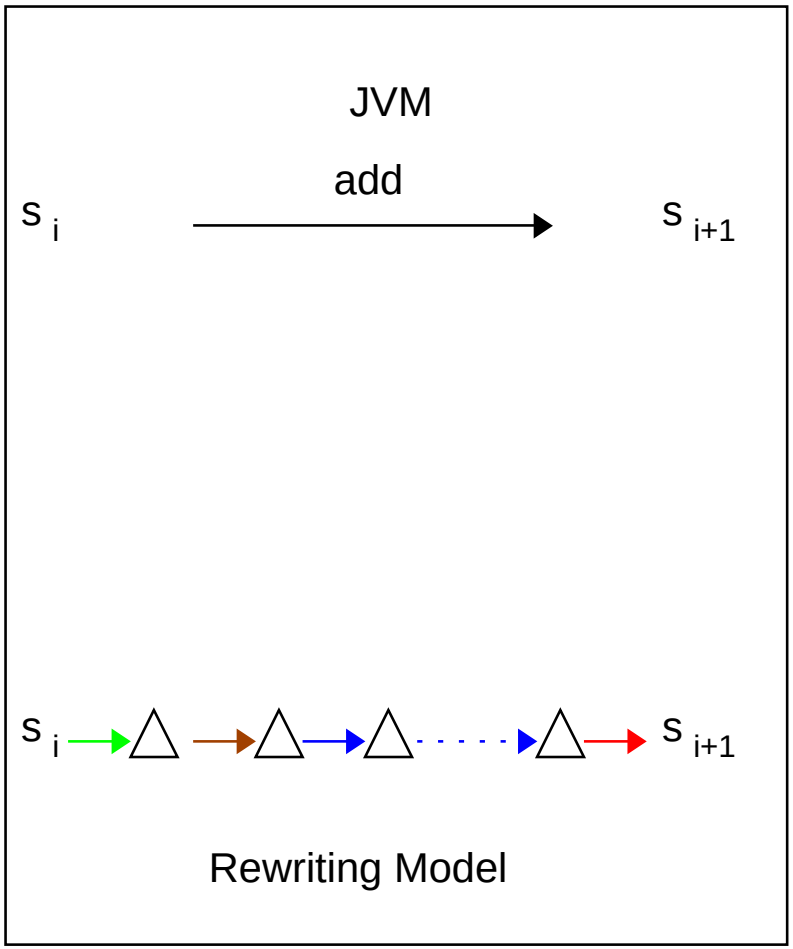
$frame(name(foo, A), pp2, s, l) \rightarrow$
$xframe(add, name(foo, A), pp2, s, l)$

$xframe(add, m, pc, stack(b, stack(a, s)), l) \rightarrow$
$xframe(xadd(a, b), m, pc, s, l)$

$xadd(succ(a), b) \rightarrow xadd(a, succ(b))$
$xadd(zero, b) \rightarrow result(b)$

$xframe(result(x), m, pc, s, l) \rightarrow$
$frame(m, next(pc), stack(x, s), l)$

Transitions JVM – Règles de réécriture



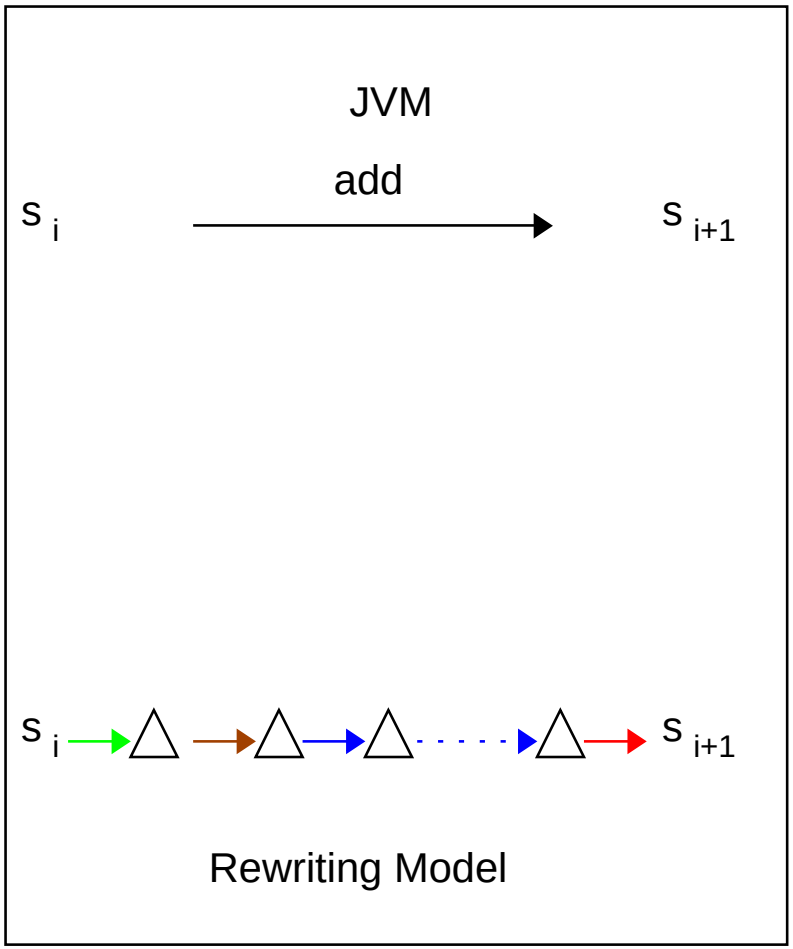
$frame(name(foo, A), pp2, s, l) \rightarrow$
$xframe(add, name(foo, A), pp2, s, l)$

$xframe(add, m, pc, stack(b, stack(a, s)), l) \rightarrow$
$xframe(xadd(a, b), m, pc, s, l)$

$xadd(succ(a), b) \rightarrow xadd(a, succ(b))$
$xadd(zero, b) \rightarrow result(b)$

$xframe(result(x), m, pc, s, l) \rightarrow$
$frame(m, next(pc), stack(x, s), l)$

Transitions JVM – Règles de réécriture



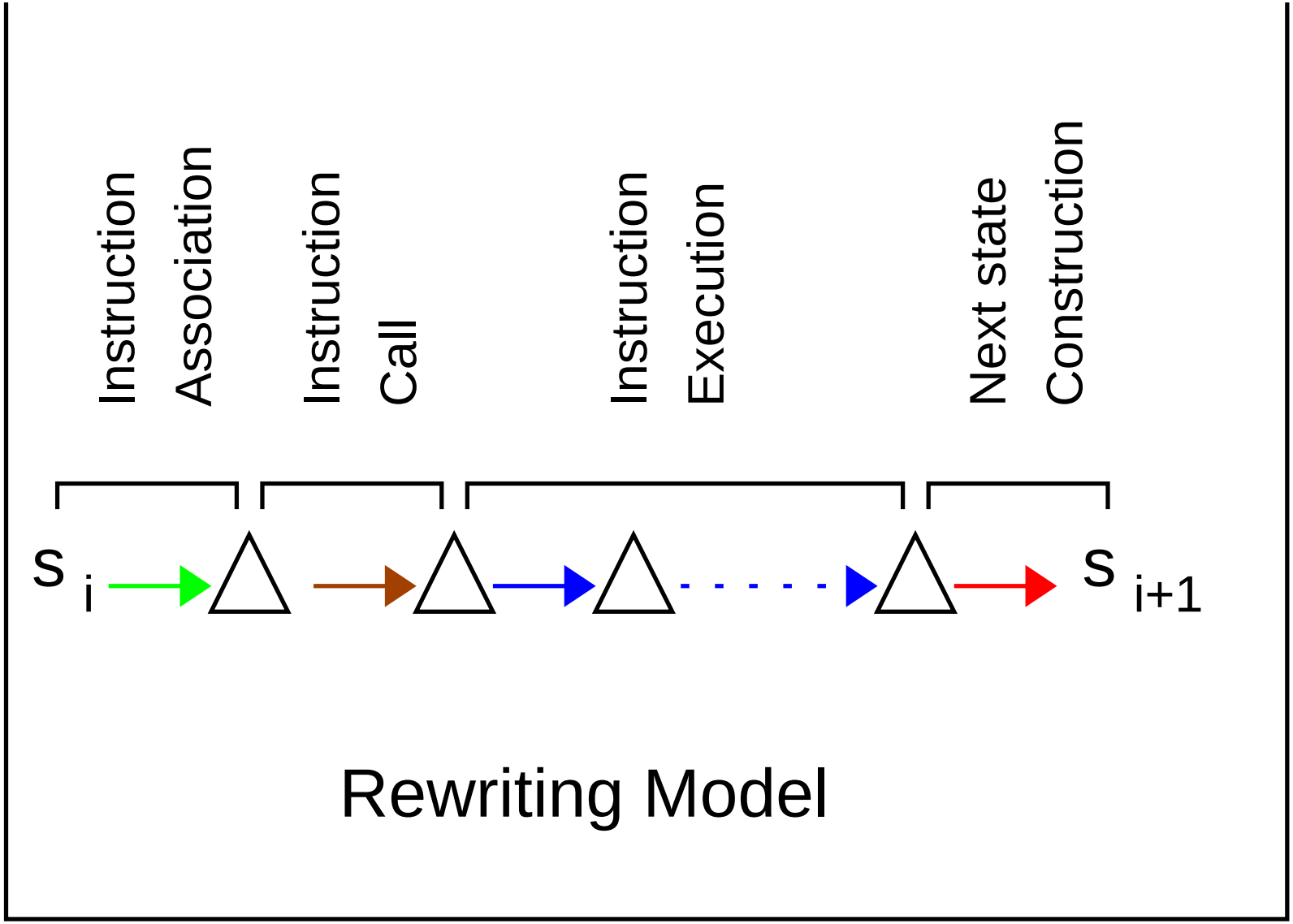
$frame(name(foo, A), pp2, s, l) \rightarrow$
$xframe(add, name(foo, A), pp2, s, l)$

$xframe(add, m, pc, stack(b, stack(a, s)), l) \rightarrow$
$xframe(xadd(a, b), m, pc, s, l)$

$xadd(succ(a), b) \rightarrow xadd(a, succ(b))$
$xadd(zero, b) \rightarrow result(b)$

$xframe(result(x), m, pc, s, l) \rightarrow$
$frame(m, next(pc), stack(x, s), l)$

Transitions JVM – Règles de réécriture



Représentation des objets

- Objets : symboles fonctionnels n-aires où chaque argument est un champ

Représentation des objets

- Objets : symboles fonctionnels n-aires où chaque argument est un champ

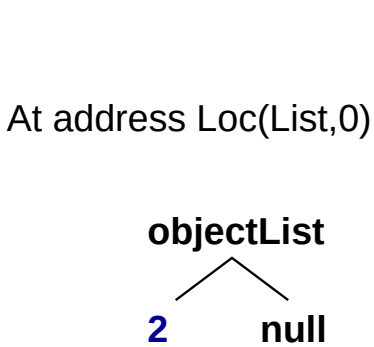
```
class List{
  int val;
  List next;
  public List(int elt,List after)
    {val=elt; next=after;}
}
...
List lpos1 = new List(2,null);
List lpos2 = new List(1,lpos1);
```

Représentation des objets

- Objets : symboles fonctionnels n-aires où chaque argument est un champ

- Création d'un objet par l'instruction *List lpos1 = new List(2,null)*:
  1. Un nouvel objet est créé et stocké dans le tas à l'adresse *loc(List,0)*
  2. Adresse stockée dans *lpos1*.

```
class List{
  int val;
  List next;
  public List(int elt,List after)
    {val=elt; next=after;}
}
...
List lpos1 = new List(2,null);
List lpos2 = new List(1,lpos1);
```



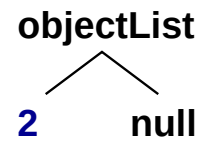
Représentation des objets

- Objets : symboles fonctionnels n-aires où chaque argument est un champ

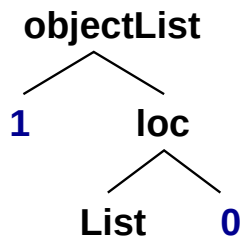
- Création d'un objet par l'instruction *List lpos1 = new List(2,null)*:
  1. Un nouvel objet est créé et stocké dans le tas à l'adresse *loc(List,0)*
  2. Adresse stockée dans *lpos1*.
- Adresse stockée dans *lpos2* est *loc(List,1)*.

```
class List{
  int val;
  List next;
  public List(int elt,List after)
    {val=elt; next=after;}
}
...
List lpos1 = new List(2,null);
List lpos2 = new List(1,lpos1);
```

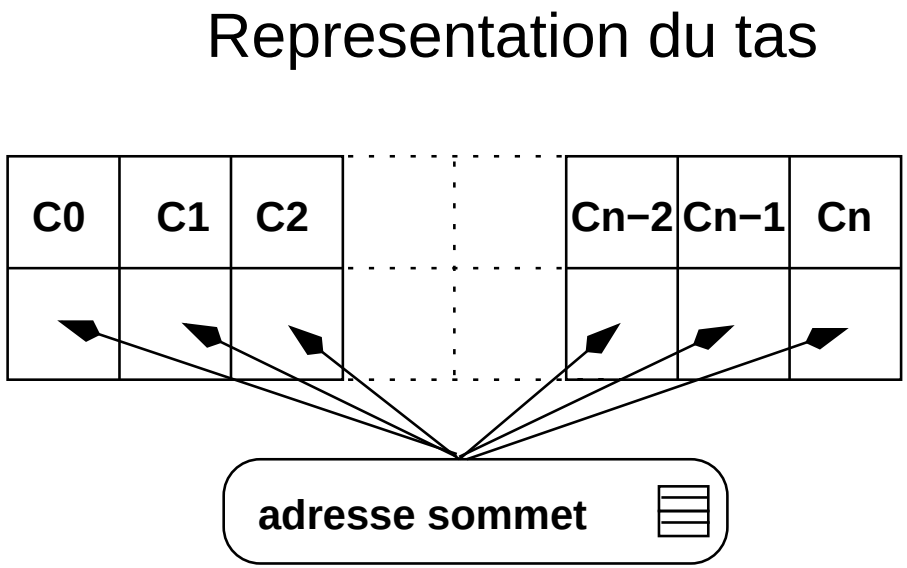
At address Loc(List,0)



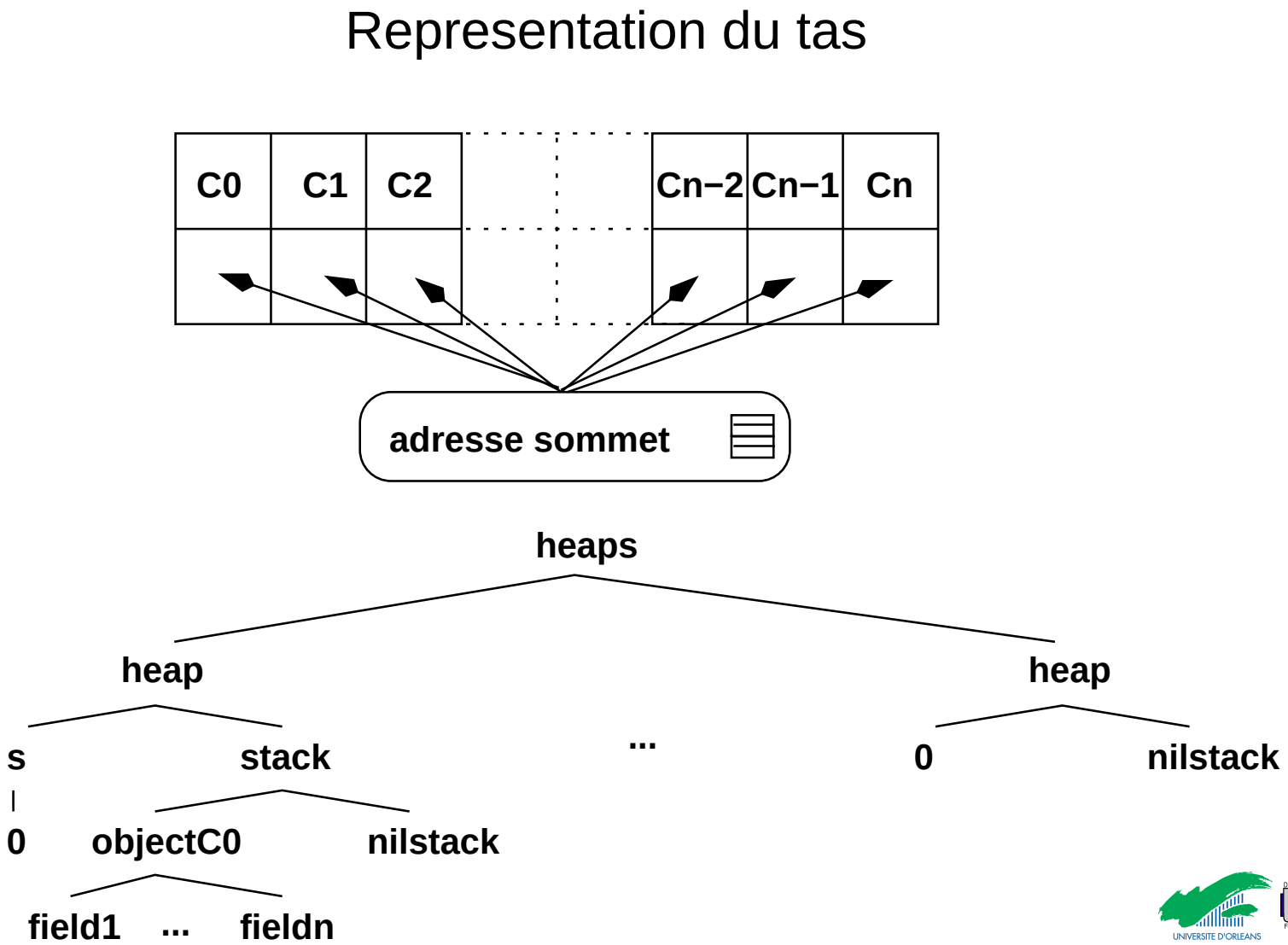
At address Loc(List,1)



Représentation du tas

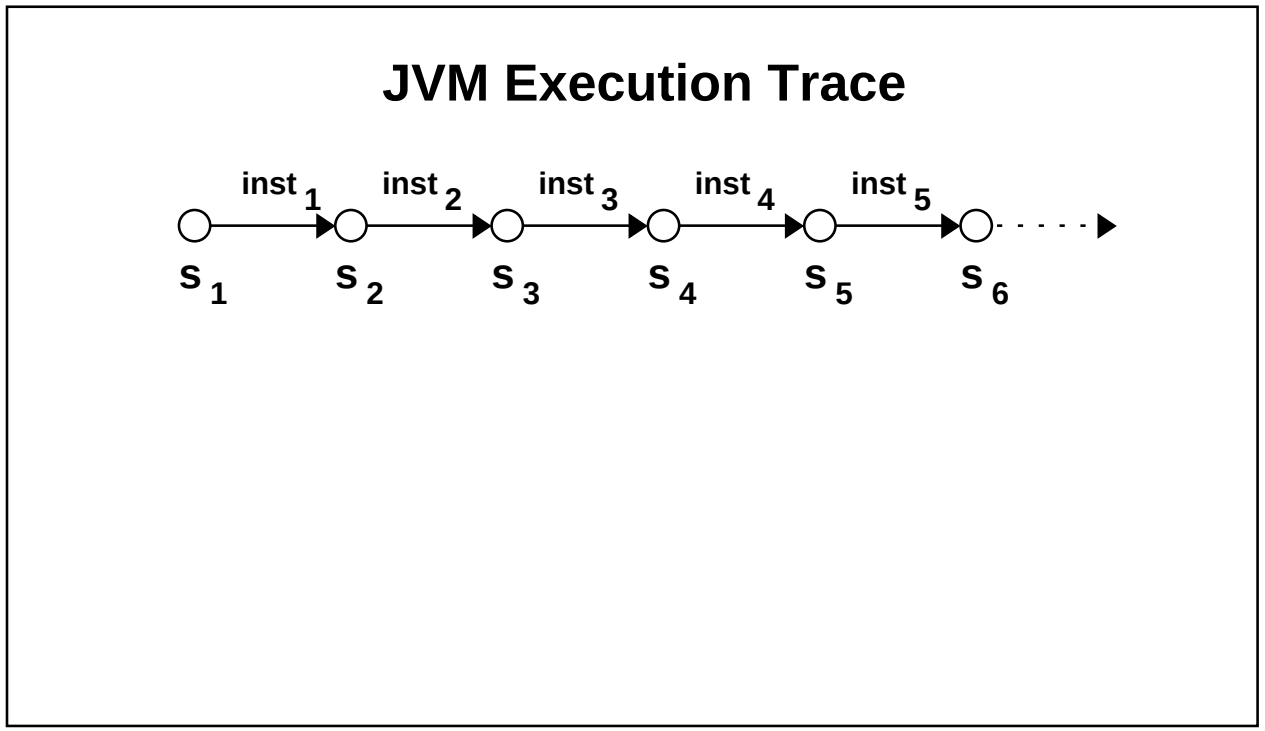


Représentation du tas



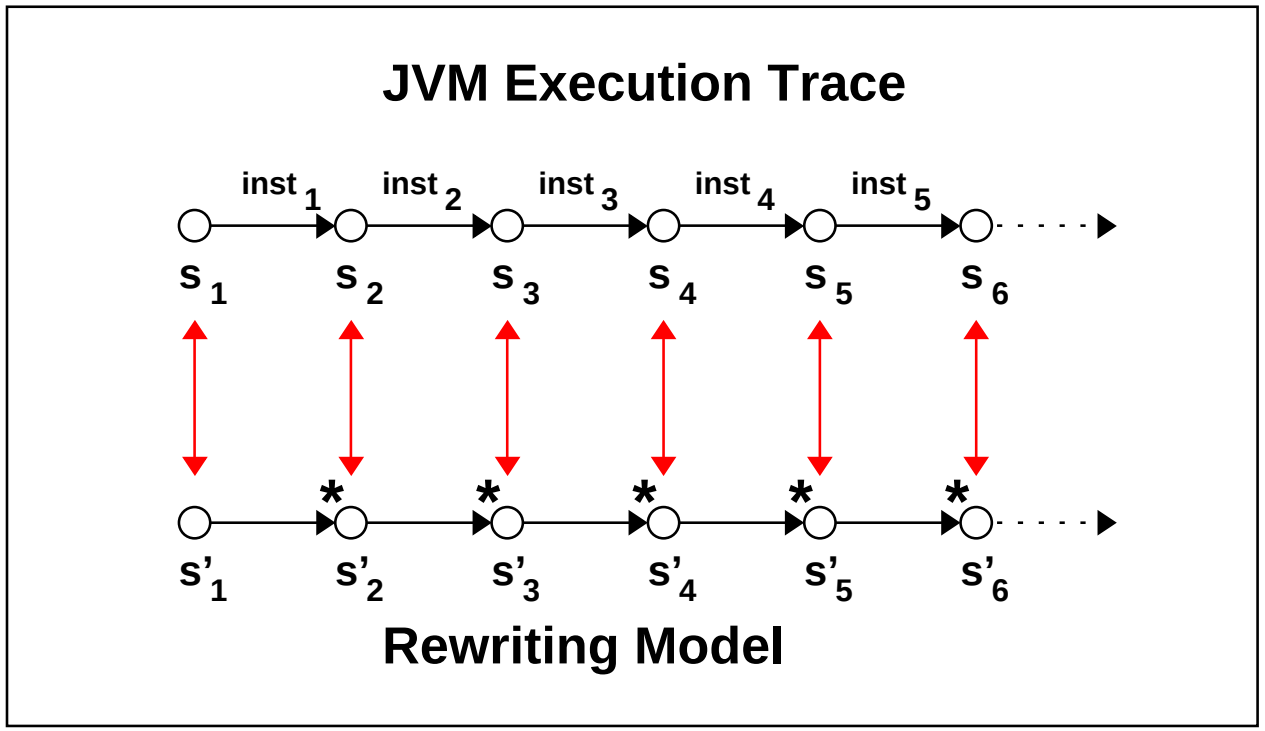
Couverture des éxécutions de la JVM

Conséquence : Chaque trace d'éxécution est couverte par un chemin de réécriture

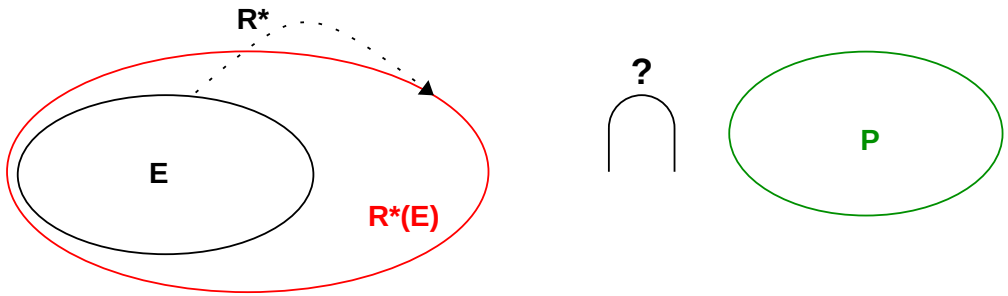


Couverture des éxécutions de la JVM

Conséquence : Chaque trace d'éxécution est couverte par un chemin de réécriture



Analyse du bytecode Java

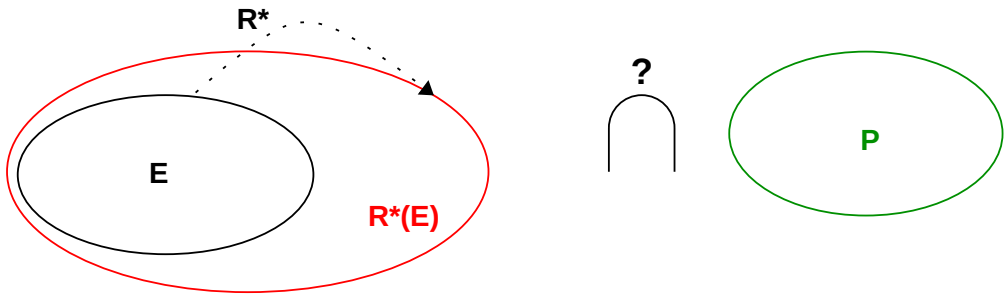


E : termes initiaux

R : système de réécriture

P : termes cibles

Analyse du bytecode Java



E : termes initiaux ⟶ **Configuration initiale de la JVM**

R : système de réécriture ⟶ **Evolution JVM + bytecode**

P : termes cibles ⟶ **Configurations interdites**

Outline

Ce qu'il faut retenir...

Ce qu'il faut retenir...

- Les systèmes de réécriture sont Turing-complets

Ce qu'il faut retenir...

- Les systèmes de réécriture sont Turing-complets
- Semi-algorithme : exploration de l'espace de recherche jusqu'à trouver un terme interdit

Ce qu'il faut retenir...

- Les systèmes de réécriture sont Turing-complets
- Semi-algorithme : exploration de l'espace de recherche jusqu'à trouver un terme interdit
  1. Si l'espace de recherche est fini, alors pas de soucis

Ce qu'il faut retenir...

- Les systèmes de réécriture sont Turing-complets
- Semi-algorithme : exploration de l'espace de recherche jusqu'à trouver un terme interdit
  1. Si l'espace de recherche est fini, alors pas de soucis
  2. Dans le cas contraire, il faut croiser les doigts pour que le terme interdit soit atteignable.

Ce qu'il faut retenir...

- Les systèmes de réécriture sont Turing-complets
- Semi-algorithme : exploration de l'espace de recherche jusqu'à trouver un terme interdit
  1. Si l'espace de recherche est fini, alors pas de soucis
  2. Dans le cas contraire, il faut croiser les doigts pour que le terme interdit soit atteignable.
  3. Si le terme interdit n'est pas atteignable alors techniques d'approximations (Vendredi après-midi)

Ce qu'il faut retenir...

- Les systèmes de réécriture sont Turing-complets
- Semi-algorithme : exploration de l'espace de recherche jusqu'à trouver un terme interdit
  1. Si l'espace de recherche est fini, alors pas de soucis
  2. Dans le cas contraire, il faut croiser les doigts pour que le terme interdit soit atteignable.
  3. Si le terme interdit n'est pas atteignable alors techniques d'approximations (Vendredi après-midi)
- Des outils de réécriture : Maude, Tom, Timbuk, ACTAS, ...

Au menu des deux dernières séances

Au menu des deux dernières séances

- Le matin : Vérification du système clien, marchand, banque avec Tom

Au menu des deux dernières séances

- Le matin : Vérification du système clien, marchand, banque avec Tom
- L'après-midi :

Au menu des deux dernières séances

- Le matin : Vérification du système clien, marchand, banque avec Tom
- L'après-midi :
  1. Technique de sur-approximation en réécriture pour la preuve de non-atteignabilité

Au menu des deux dernières séances

- Le matin : Vérification du système clien, marchand, banque avec Tom
- L'après-midi :
  1. Technique de sur-approximation en réécriture pour la preuve de non-atteignabilité
  2. Spécification d'un système infini : un système de réécriture et un automate d'arbres

Au menu des deux dernières séances

- Le matin : Vérification du système clien, marchand, banque avec Tom
- L'après-midi :
  1. Technique de sur-approximation en réécriture pour la preuve de non-atteignabilité
  2. Spécification d'un système infini : un système de réécriture et un automate d'arbres
  3. Spécification de la propriété à vérifier : codage de sa négation par un automate d'arbres

Au menu des deux dernières séances

- Le matin : Vérification du système clien, marchand, banque avec Tom
- L'après-midi :
  1. Technique de sur-approximation en réécriture pour la preuve de non-atteignabilité
  2. Spécification d'un système infini : un système de réécriture et un automate d'arbres
  3. Spécification de la propriété à vérifier : codage de sa négation par un automate d'arbres
  4. Calcule de la sur-approximation et preuve de la propriété

Au menu des deux dernières séances

- Le matin : Vérification du système clien, marchand, banque avec Tom
- L'après-midi :
  1. Technique de sur-approximation en réécriture pour la preuve de non-atteignabilité
  2. Spécification d'un système infini : un système de réécriture et un automate d'arbres
  3. Spécification de la propriété à vérifier : codage de sa négation par un automate d'arbres
  4. Calcule de la sur-approximation et preuve de la propriété

Installez Tom : http://tom.loria.fr