Vérification de Code d'Octet de la Machine Virtuelle Java

Formalisation & Implantation

Eva Rose

Soutenance de Thèse

UFR d'Informatique
Université Paris 7 – Denis Diderot

Le 27 septembre 2002

Ecole Normale Supérieure de Cachan

Cadre

Problème

Conception

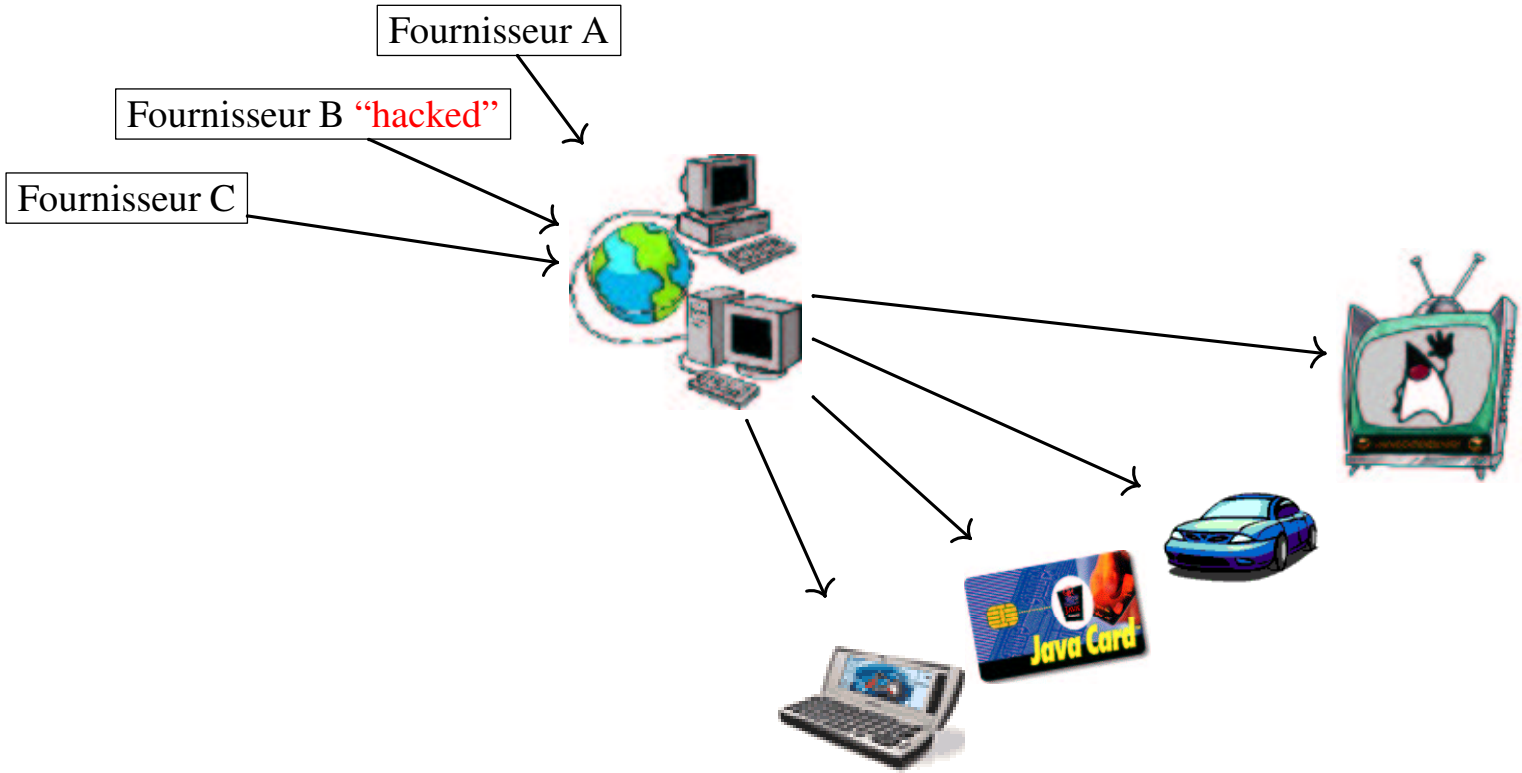
Formalisation

Réalisation

Conclusion

Exemples

Contexte


Fournisseur A
Fournisseur B "hacked"
Fournisseur C
Java Card

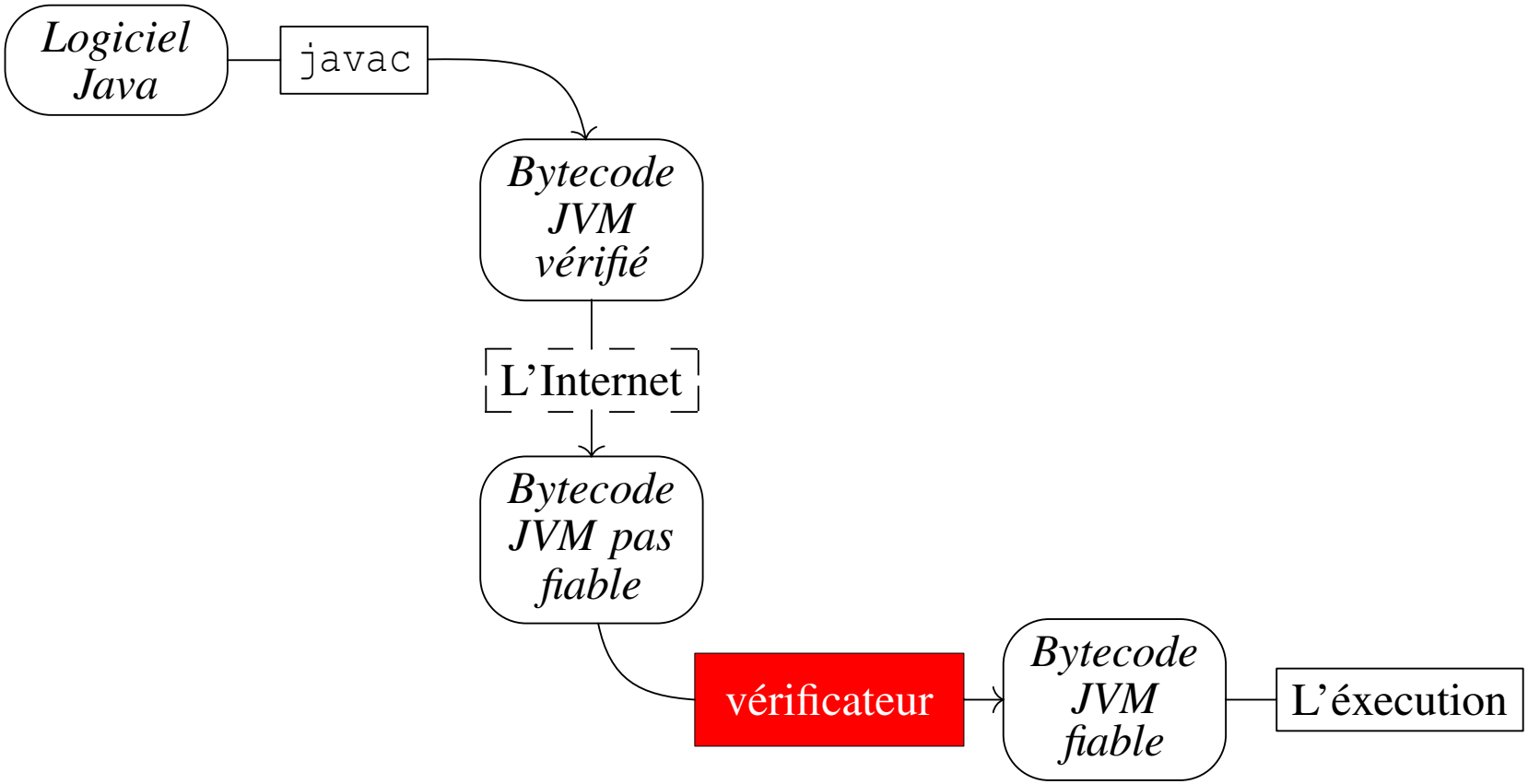
Cadre

Situation: code binaire Java est chargé par un réseau *peu fiable* et envoyé vers des systèmes Java *imbriqués*, restreints en mémoire.

Vérification de typage: un composant de base pour la sécurité.

Souhaite: effectuer la vérification de typage de code binaire Java *directement* sur les plate-formes cibles du code.

Vérification standard du typage de bytecode



Modèle de mémoire

En générale notre plate-forme ciblée contient la mémoire RAM **scratch** et RAM **flash**.

Scratch: lire et écrire chaque octet directement. *Chère*.
Taille typique: $1-2$KB d'octets (Java Card 2001).

Flash: lire par octet, mais écrire en block. *Généralement bon marché*.
Taille typique: $64-128$KB d'octets (Java Card 2001).

ROM/EEPROM: lire par octet/écrire une nombre de fois fixé. *Obsolète*.

Vérificateur standard typiquement 50KB mémoire de flash pour le logiciel (binary), $30-100$KB mémoire de scratch.

Cadre

Problème

Conception

Formalisation

Réalisation

Conclusion

Exemples

Le problème

Effectuer la vérification de typage de code binaire Java sur les systèmes restreint en mémoire.

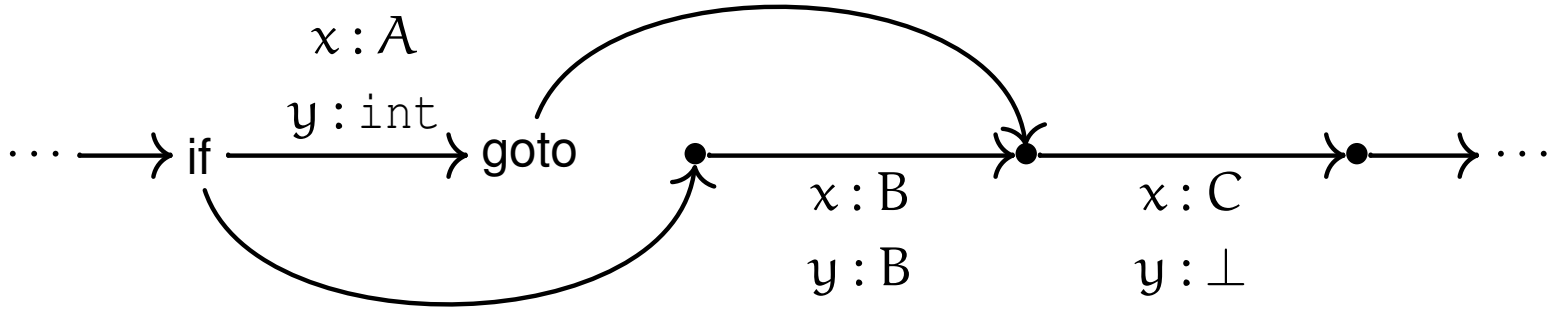
La sûretée de typage Java

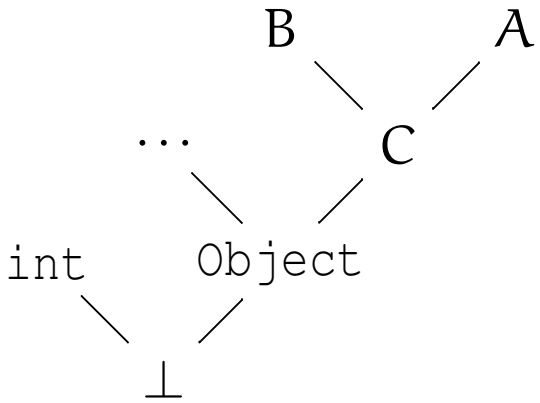
```
class Q {...
  void someMethod(T x) { ... x.dummy(); ... }
...}

   ... someMethod(new S()); ...
```

Pour que l'invocation de x.dummy() réussisse, il faut que le contraint statique : "S **extends** ... **extends** T" s'applique, formellement.

"LES METHODES BIEN TYPEES REUSSIRONT L'EXECUTION"

Flôt de données dans un language d'objets





Un exemple d'une vérification standard

```
class B {int dummy() {return 0;}}
class A extends B{}

class E {
  void m() {
    B x = new A();
    while (x.dummy() != 0)
      x = new B();
  }
}
```

```
OK      <*, E.bot>  0: new A  <<<
          top       7: astore 1
          top       8: goto 19
          top      11: new B
          top      18: astore 1
          top      19: aload 1
          top      20: invokevirtual mref(B,msig(dummy,[]),I)
          top      23: ifne 11
          top      26: return
          top
```

```
OK      <*, E.bot>  0: new A
OK      <A, E.bot>  7: astore 1  <<<
          top       8: goto 19
          top      11: new B
          top      18: astore 1
          top      19: aload 1
          top      20: invokevirtual mref(B,msig(dummy,[]),I)
          top      23: ifne 11
          top      26: return
          top
```

```
OK      <*, E.bot>  0: new A
OK      <A, E.bot>  7: astore 1
OK      <*, E.A>    8: goto 19   <<<
          top      11: new B
          top      18: astore 1
          top      19: aload 1
          top      20: invokevirtual mref(B,msig(dummy,[]),I)
          top      23: ifne 11
          top      26: return
          top
```

```
OK      <*, E.bot>   0: new A
OK      <A, E.bot>   7: astore 1
OK      <*, E.A>     8: goto 19
          top       11: new B
          top       18: astore 1
OK      <*, E.A>    19: aload 1  <<<
          top       20: invokevirtual mref(B,msig(dummy,[]),I)
          top       23: ifne 11
          top       26: return
          top
```

```
OK      <*, E.bot>  0: new A
OK      <A, E.bot>  7: astore 1
OK      <*, E.A>    8: goto 19
          top      11: new B
          top      18: astore 1
OK       <*, E.A>  19: aload 1
OK       <A, E.A>  20: invokevirtual mref(B,msig(dummy,[]),I)  <<<
          top      23: ifne 11
          top      26: return
          top
```

```
OK      <*, E.bot>   0: new A
OK      <A, E.bot>   7: astore 1
OK      <*, E.A>     8: goto 19
          top       11: new B
          top       18: astore 1
OK      <*, E.A>    19: aload 1
OK      <A, E.A>    20: invokevirtual mref(B,msig(dummy,[]),I)
OK      <I, E.A>    23: ifne 11  <<<
          top       26: return
          top
```

```
OK      <*, E.bot>  0: new A
OK      <A, E.bot>  7: astore 1
OK      <*, E.A>    8: goto 19
OK      <*, E.A>   11: new B  <<<
          top      18: astore 1
OK      <*, E.A>   19: aload 1
OK      <A, E.A>   20: invokevirtual mref(B,msig(dummy,[]),I)
OK      <I, E.A>   23: ifne 11
OK      <*, E.A>   26: return  <<<
          top
```

```
OK    <*, E.bot>   0: new A
OK    <A, E.bot>   7: astore 1
OK    <*, E.A>     8: goto 19
OK    <*, E.A>    11: new B  <<<
        top       18: astore 1
OK    <*, E.A>    19: aload 1
OK    <A, E.A>    20: invokevirtual mref(B,msig(dummy,[]),I)
OK    <I, E.A>    23: ifne 11
OK    <*, E.A>    26: return
OK      top
```

```
OK    <*, E.bot>   0: new A
OK    <A, E.bot>   7: astore 1
OK    <*, E.A>     8: goto 19
OK    <*, E.A>    11: new B
OK    <B, E.A>    18: astore 1  <<<
OK    <*, E.A>    19: aload 1
OK    <A, E.A>    20: invokevirtual mref(B,msig(dummy,[]),I)
OK    <I, E.A>    23: ifne 11
OK    <*, E.A>    26: return
OK      top
```

```
OK    <*, E.bot>   0: new A
OK    <A, E.bot>   7: astore 1
OK    <*, E.A>     8: goto 19
      <*, E.A>    11: new B
      <B, E.A>    18: astore 1
OK    <*, E.B>    19: aload 1  <<<
      <A, E.A>    20: invokevirtual mref(B,msig(dummy,[]),I)
      <I, E.A>    23: ifne 11
      <*, E.A>    26: return
OK    top
```

```
OK    <*, E.bot>   0: new A
OK    <A, E.bot>   7: astore 1
OK    <*, E.A>     8: goto 19
      <*, E.A>    11: new B
      <B, E.A>    18: astore 1
OK    <*, E.B>    19: aload 1
OK    <B, E.B>    20: invokevirtual mref(B,msig(dummy,[]),I)  <<<
      <I, E.A>    23: ifne 11
      <*, E.A>    26: return
OK      top
```

```
OK    <*, E.bot>   0: new A
OK    <A, E.bot>   7: astore 1
OK    <*, E.A>     8: goto 19
      <*, E.A>    11: new B
      <B, E.A>    18: astore 1
OK    <*, E.B>    19: aload 1
OK    <B, E.B>    20: invokevirtual mref(B,msig(dummy,[]),I)
OK    <I, E.B>    23: ifne 11  <<<
      <*, E.A>    26: return
OK      top
```

```
OK    <*, E.bot>   0: new A
OK    <A, E.bot>   7: astore 1
OK    <*, E.A>     8: goto 19
OK    <*, E.B>     11: new B  <<<
      <B, E.A>     18: astore 1
OK    <*, E.B>     19: aload 1
OK    <B, E.B>     20: invokevirtual mref(B,msig(dummy,[]),I)
OK    <I, E.B>     23: ifne 11
OK    <*, E.B>     26: return  <<<
OK      top
```

```
OK    <*, E.bot>    0: new A
OK    <A, E.bot>    7: astore 1
OK    <*, E.A>      8: goto 19
OK    <*, E.B>     11: new B  <<<
      <B, E.A>     18: astore 1
OK    <*, E.B>     19: aload 1
OK    <B, E.B>     20: invokevirtual mref(B,msig(dummy,[]),I)
OK    <I, E.B>     23: ifne 11
OK    <*, E.B>     26: return
OK      top
```

```
OK    <*, E.bot>    0: new A
OK    <A, E.bot>    7: astore 1
OK    <*, E.A>      8: goto 19
OK    <*, E.B>     11: new B
OK    <B, E.B>     18: astore 1  <<<
OK    <*, E.B>     19: aload 1
OK    <B, E.B>     20: invokevirtual mref(B,msig(dummy,[]),I)
OK    <I, E.B>     23: ifne 11
OK    <*, E.B>     26: return
OK      top
```

```
OK     <*, E.bot>     0: new A
OK     <A, E.bot>     7: astore 1
OK     <*, E.A >      8: goto +11
OK     <*, E.B>      11: new B
OK     <B, E.B>      18: astore 1
OK     <*, E.B>      19: aload 1
OK     <B, E.B>      20: invokevirtual mref(B,msig(dummy,[]),I)
OK     <I, E.B>      23: ifne -12
OK     <*, E.B>      26: return
OK       top
```

Cadre

Problème

Conception

Formalisation

Réalisation

Conclusion

Exemples

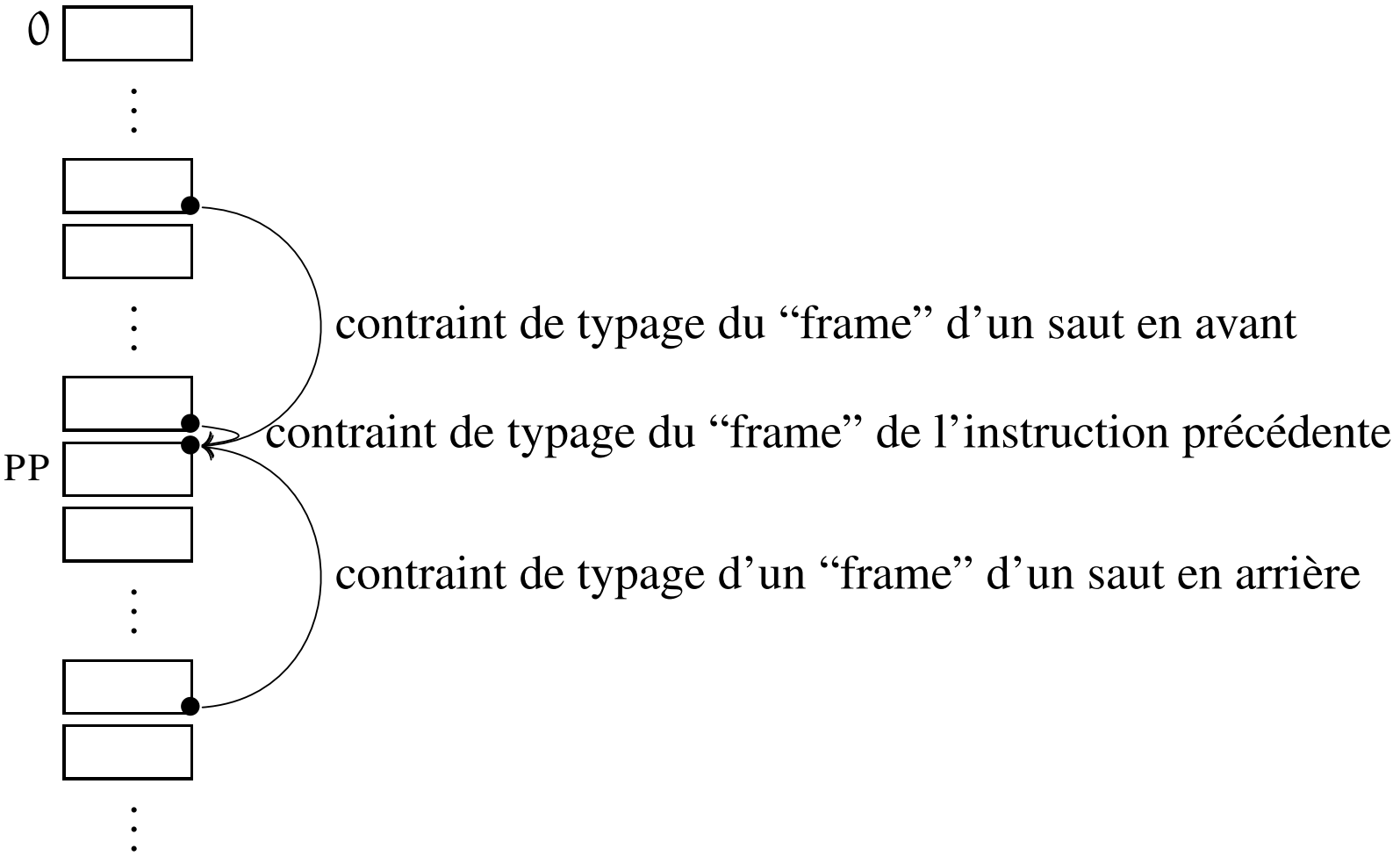
Stratégie

L'idée: application de "certificat" pour vérifier la sûretée de typage du code binaire en plusieurs étapes où la dernière est un simple *parcour en avant*. Ceci inspiré par PCC ("Proof Carrying Code").

Formalisation: pour sécuriser l'implantation contre les erreurs de type théorique. (Ceci est surtout important pour les systèmes imbriqués, distribués en grand nombres, qui portent les informations sensibles.)

Implantation de prototype: pour illustrer le concept en pratique, et pour supporter un modèle à tester.

Contraints de typage possible dans le code



Vérification du code parcouru en avant

Saut en avant. Le contraint de typage est vérifié au *but du saut* PP. Nous retardons la vérification en enregistrant le contraint ("pending") pendant le traitement du saut.

Saut en arrière. Le contraint de typage est vérifié à la *source du saut*. En conséquence,

1. il faut que le contraint de typage soit assuré ("saved") à PP pour être disponible au moment où nous traiterons le saut à sa source, et
2. le contraint de typage peut être insuffisant pour effectuer une vérification.

Vérification du code parcouru en avant

L'information externe nécessaire pour cette vérification (le certificat):

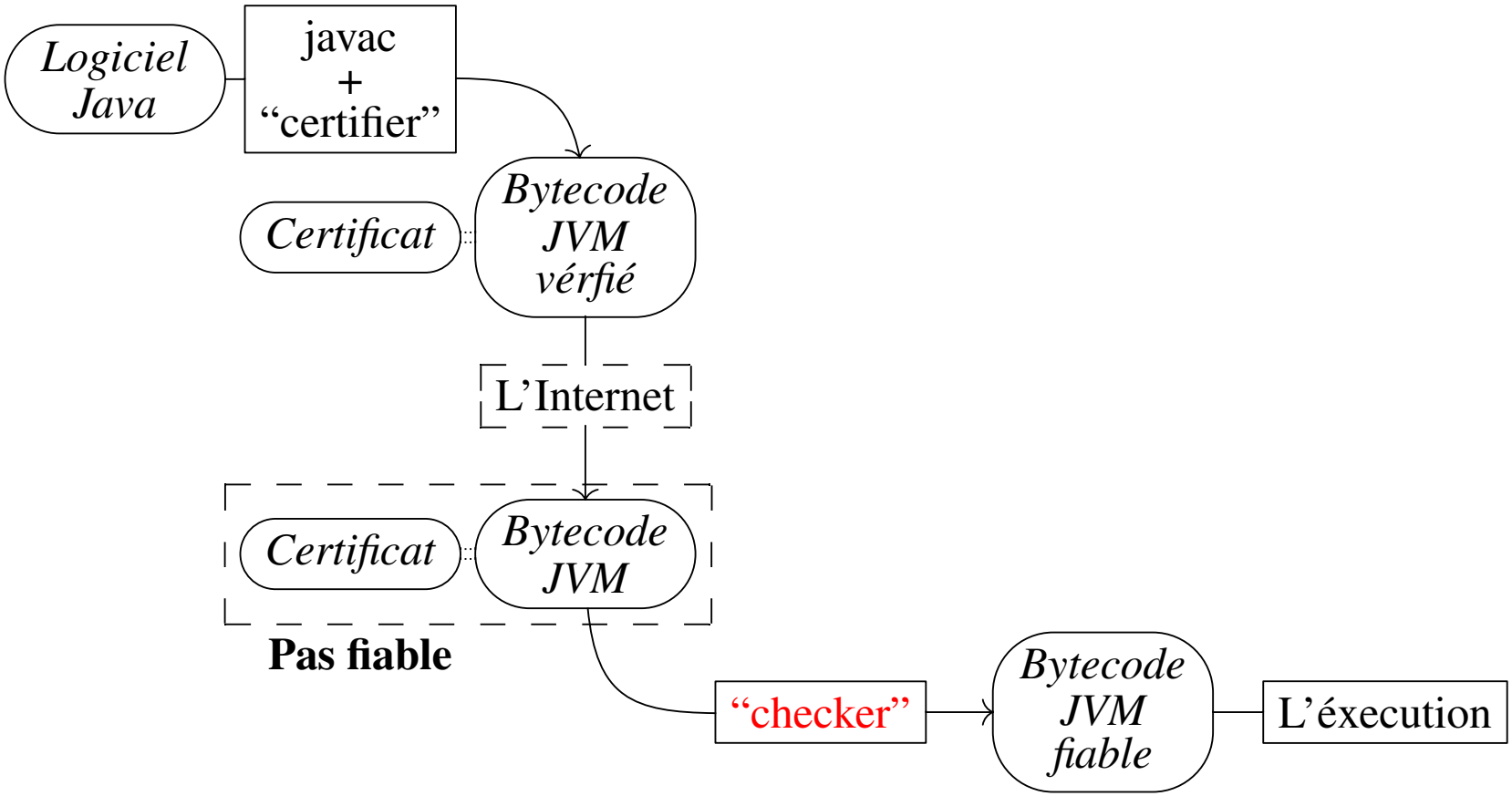
- Contraints de typage pour les situations incomplètes (les *frame type certificates*).
- Les positions des buts des sauts en arrière (les *labels*).

Information nécessaire pour l'exemple

<*, E.B>

```
 0: new LA;
 7: astore 1
 8: goto +11
11: new B
18: astore 1
19: aload 1
20: invokevirtual mref(B,msig(dummy,[]),I)
23: ifne -12
26: return
```

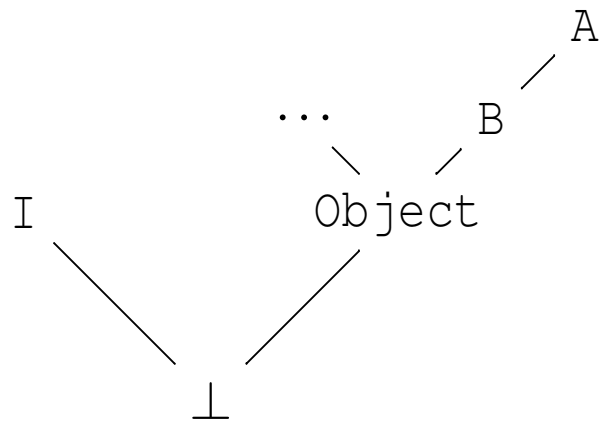
Vérification *Lightweight* du typage de bytecode



Un exemple d'une vérification *lightweight*

```
class B {int dummy() {return 0;}}
class A extends B{}

class E {
  void m() {
    B x = new A();
    while (x.dummy() != 0)
      x = new B();
  }
}
```

Sa hiérarchie des types



```
OK   <*, E.bot>     0: new A   <<<
                    7: astore 1
                    8: goto 19
                   11: new B
                   18: astore 1
                   19: aload 1
                   20: invokevirtual mref(B,msig(dummy,[]),I)
                   23: ifne 11
                   26: return
```

$$\mathrm{CE} = \langle \{11\}, \{11 \mapsto \texttt{<*, E.bot>}\} \rangle$$

$$\mathrm{P} = \{\}$$

$$\mathrm{S} = \{\}$$

```
OK                    0: new A
OK   <A, E.bot>       7: astore 1   <<<
                      8: goto 19
                     11: new B
                     18: astore 1
                     19: aload 1
                     20: invokevirtual mref(B,msig(dummy,[]),I)
                     23: ifne 11
                     26: return
```

$$\mathrm{CE} = \langle \{11\}, \{11 \mapsto \texttt{<*, E.bot>}\} \rangle$$

$$\mathrm{P} = \{\}$$

$$\mathrm{S} = \{\}$$

```
OK                    0: new A
OK                    7: astore 1
OK   <*, E.A>         8: goto 19   <<<
                     11: new B
                     18: astore 1
                     19: aload 1
                     20: invokevirtual mref(B,msig(dummy,[]),I)
                     23: ifne 11
                     26: return
```

$$\mathrm{CE} = \langle \{11\}, \{11 \mapsto \texttt{<*, E.bot>}\} \rangle$$

$$\mathrm{P} = \{19 \mapsto \texttt{<*, E.A>}\}$$

$$\mathrm{S} = \{\}$$

```
OK                    0: new A
OK                    7: astore 1
OK                    8: goto 19
OK   <*, E.bot>      11: new B    <<<
                     18: astore 1
                     19: aload 1
                     20: invokevirtual mref(B,msig(dummy,[]),I)
                     23: ifne 11
                     26: return
```

$$\mathrm{CE} = \langle \{11\}, \{11 \mapsto \texttt{<*, E.bot>}\} \rangle$$

$$\mathrm{P} = \{19 \mapsto \texttt{<*, E.A>}\}$$

$$\mathrm{S} = \{11 \mapsto \texttt{<*, E.bot>}\}$$

```
OK                    0: new A
OK                    7: astore 1
OK                    8: goto 19
OK                   11: new B
OK    <B, E.bot>     18: astore 1    <<<
                     19: aload 1
                     20: invokevirtual mref(B,msig(dummy,[]),I)
                     23: ifne 11
                     26: return
```

$$\mathrm{CE} = \langle \{11\}, \{11 \mapsto \texttt{<*, E.bot>}\} \rangle$$

$$\mathrm{P} = \{19 \mapsto \texttt{<*, E.A>}\}$$

$$\mathrm{S} = \{11 \mapsto \texttt{<*, E.bot>}\}$$

```
OK                  0: new A
OK                  7: astore 1
OK                  8: goto 19
OK                 11: new B
OK                 18: astore 1
OK   <*, E.B>      19: aload 1   <<<
                   20: invokevirtual mref(B,msig(dummy,[]),I)
                   23: ifne 11
                   26: return
```

$$\mathrm{CE} = \langle \{11\}, \{11 \mapsto \texttt{<*, E.bot>}\} \rangle$$

$$\mathrm{P} = \{19 \mapsto \texttt{<*, E.A>}\}$$

$$\mathrm{S} = \{11 \mapsto \texttt{<*, E.bot>}\}$$

```
OK                  0: new A
OK                  7: astore 1
OK                  8: goto 19
OK                 11: new B
OK                 18: astore 1
OK                 19: aload 1
OK   <B, E.B>      20: invokevirtual mref(B,msig(dummy,[]),I)    <<<
                   23: ifne 11
                   26: return
```

$$\mathrm{CE} = \langle \{11\}, \{11 \mapsto \texttt{<*, E.bot>}\} \rangle$$

$$\mathrm{P} = \{\}$$

$$\mathrm{S} = \{11 \mapsto \texttt{<*, E.bot>}\}$$

```
OK                  0: new A
OK                  7: astore 1
OK                  8: goto 19
OK                 11: new B
OK                 18: astore 1
OK                 19: aload 1
OK                 20: invokevirtual mref(B,msig(dummy,[]),I)
OK   <I, E.B>      23: ifne 11   <<<
                   26: return
```

$$\mathrm{CE} = \langle \{11\}, \{11 \mapsto \texttt{<*, E.bot>}\} \rangle$$

$$\mathrm{P} = \{\}$$

$$\mathrm{S} = \{11 \mapsto \texttt{<*, E.bot>}\}$$

```
OK                    0: new A
OK                    7: astore 1
OK                    8: goto 19
OK                   11: new B
OK                   18: astore 1
OK                   19: aload 1
OK                   20: invokevirtual mref(B,msig(dummy,[]),I)
OK                   23: ifne 11
OK   <*, E.B>        26: return   <<<
```

$$\mathrm{CE} = \langle \{11\}, \{11 \mapsto \texttt{<*, E.bot>}\} \rangle$$

$$\mathrm{P} = \{\}$$

$$\mathrm{S} = \{11 \mapsto \texttt{<*, E.bot>}\}$$

```
OK                  0: new A
OK                  7: astore 1
OK                  8: goto 19
OK                 11: new B
OK                 18: astore 1
OK                 19: aload 1
OK                 20: invokevirtual mref(B,msig(dummy,[]),I)
OK                 23: ifne 11
OK                 26: return
OK   top
```

$$\mathrm{CE} = \langle \{11\}, \{11 \mapsto \texttt{<*, E.bot>}\} \rangle$$

$$\mathrm{P} = \{\}$$

$$\mathrm{S} = \{11 \mapsto \texttt{<*, E.bot>}\}$$

Utilisation de mémoire

$$\begin{aligned}\#\text{type descriptors} &= (\text{MS} + \text{ML}) \times (1 + \#\text{LS} + q) \\ &= (1+2) \times (1+1+1) = 6\end{aligned}$$

où #LS est le nombre total des sauts en arrière, et q est le nombre le plus grand des sauts simultanés en avant.

Cadre

Problème

Conception

Formalisation

Réalisation

Conclusion

Exemples

Le sous-language considéré

Nous considérons un sous-ensemble de la machine virtuelle suffissant pour compiler un sous-language non trivial qui notament permet

- la créeation et manipulation des objets,
- l'appel des méthodes d'objets, et
- l'access aux variables d'objets.

Egalement nous voulons permetre d'écrire les programme contenant les boucles, les branchements, ou la récursion, et les esxceptions.

La machine virtuelle considéré

Sous ensemble des instructions:

iconst_0, iconst_1, aconst_null, dup, pop, iadd, isub,

istore, astore, iload, aload,

iastore, aastore, iaload, aaload, newarray, anewarray, arraylength,

checkcast, ldc_w, new, getfield, putfield, invokevirtual,

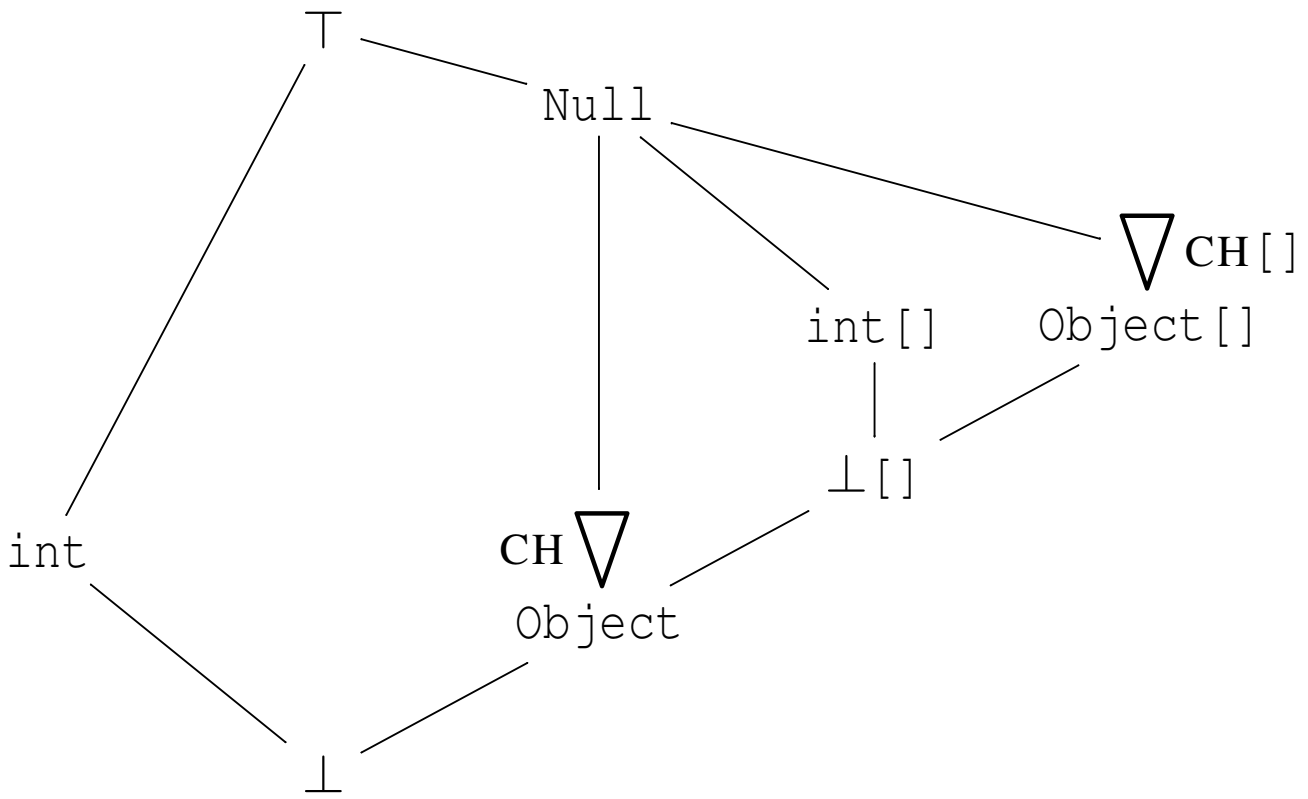
ifne, ifle, ifnull, goto,

athrow, return, ireturn, areturn

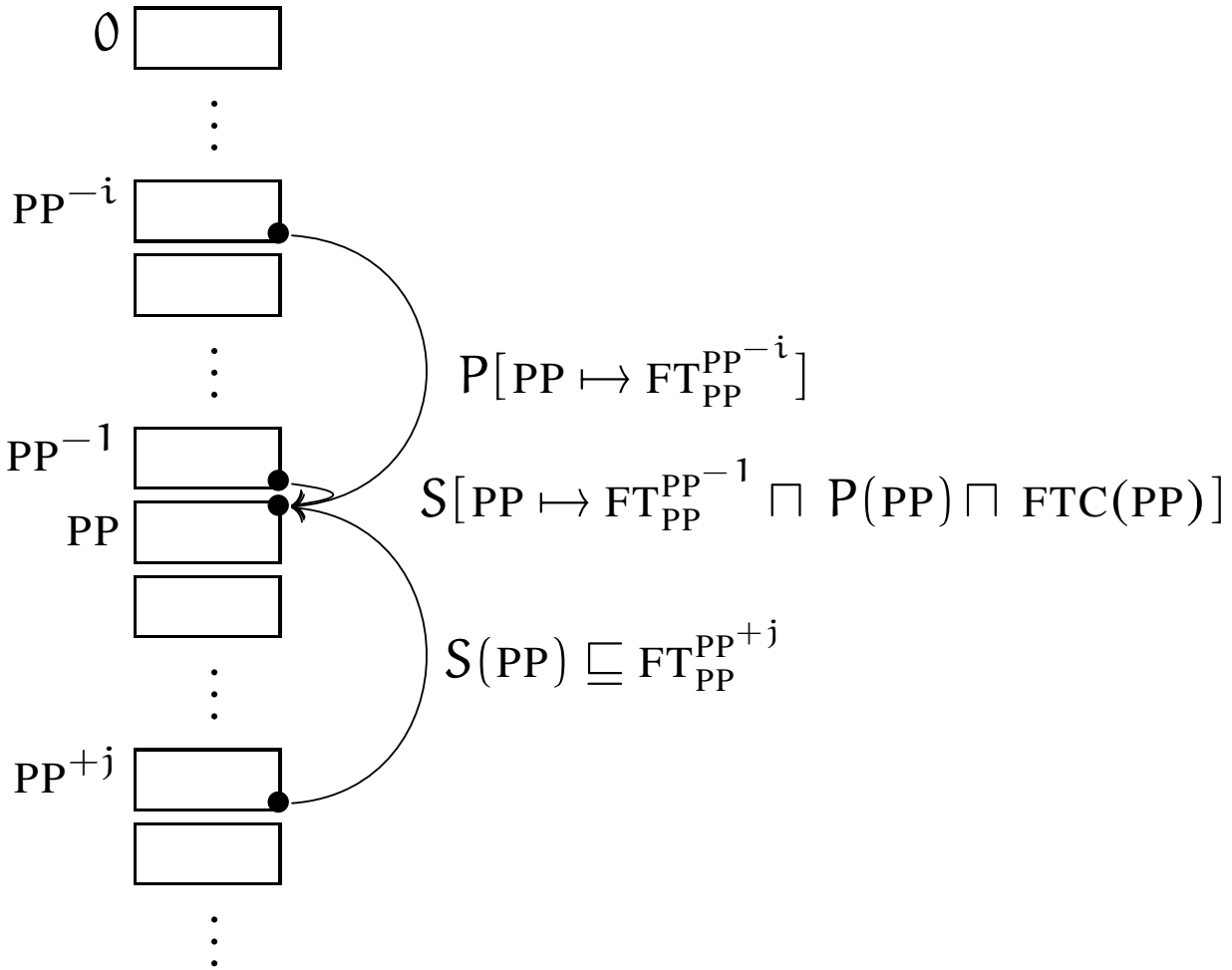
Types de base: `int`.

Types de reference: class reference types `TypeName`, *ou* one-dimensional arrays of class reference types `TypeName[]`, *ou* `int[]`.

L'ordre de la sûrtée des types pour l'ensemble des instructions considerée



Reconstruction d'une solution ("lightweight")



La construction d'un certificat

Soit FTA une solution pour l'ensemble des contraintes du typage d'une méthode.

1. Suppose qu'il y a un saut de PP à PP'' dans le code. Si $\text{PP}'' \leq \text{PP}$ (saut en arrière) alors PP'' est placé dans le certificat (les labels LS).
2. Suppose qu'il y a k saut en avant de $\text{PP}^{(-i)}$ à PP dans le code. Si $\text{FTA}(\text{PP}) \sqsubset \left(\bigsqcap_{i=1}^{k} \text{P}(\text{PP})_{\text{PP}^{(-i)}} \sqcap \text{FT}_{\text{PP}}^{\text{PP}^{(-1)}}\right)$ alors la solution $\text{FTA}(\text{PP})$ est placée dans le certificat (frame types FTC).

Théorème: le vérificateur "lightweight" s'applique d'une façon fiable.

Nous montreront que pour un contexte de vérification Γ, une méthode M, avec sa solution FTA à l'ensemble des contraintes de typages de la méthode, les lemmes suivants sont équivalent:

1. Il existe une solution (frame type assignment) FTA pour M, tel que $\Gamma \vdash_{\mathrm{bv}} \mathrm{M}, \mathrm{FTA}$ est prouvable.
2. Il existe un certificat CE pour M, tel que $\Gamma \vdash_{\mathrm{lbv}} \mathrm{M}, \mathrm{CE}$ est prouvable.

Lemme principal

Le lemme (6.2.6) repète la théorème principal par une formulation qui dépend de la semantique de la construction des certificats ("certifier").

(6.2.6a) $$\Gamma \vdash_{\mathrm{bv}} \mathrm{M}, \mathrm{FTA} \Leftrightarrow \exists! \, \mathrm{CE} : \Gamma \vdash_{\mathrm{lbc}} \mathrm{M}, \mathrm{FTA}, \mathrm{CE}$$

(6.2.6b) $$\Gamma \vdash_{\mathrm{lbv}} \mathrm{M}, \mathrm{CE} \Leftrightarrow \exists! \, \mathrm{FTA} : \Gamma \vdash_{\mathrm{lbc}} \mathrm{M}, \mathrm{FTA}, \mathrm{CE}$$

Les quatre parties de la démonstration

Le cas (BV) $\Rightarrow$ (LBC) montre formellement l'existence d'un algorithme pour un *pré-vérificateur* (la première étape d'une vérification du typage).

Le cas (LBC) $\Rightarrow$ (BV) montre formellement que cet algorithme est correct par rapport à la vérification standard.

Le cas (LBV) $\Rightarrow$ (LBC) montre formellement qu'une solution FTA pour l'ensemble des contraintes de typages d'une méthode peut être reconstruit de la part d'un certificat CE de la méthode.

Le cas (LBC) $\Rightarrow$ (LBV) montre formellement que à partir d'un *pré-vérificateur* nous pouvons construire un algorithme de vérification du typage de code, uniquement basé sur un certificat CE, de la méthode.

Première expansion de l'abre sémantique.

(4.2.7a)
$$\frac{\mathrm{CH} \vdash \mathrm{FTA}(0) \sqsubseteq \mathrm{FT}_0 \qquad \Omega \vdash 0, \mathrm{C}, \emptyset \overset{\text{tsafe}}{\Rightarrow} \mathrm{PPS}}{\Gamma \vdash_{\mathrm{bv}} \mathrm{M}, \mathrm{FTA}}$$

(6.2.4a)
$$\frac{\mathrm{CH} \vdash \mathrm{FTA}(0) = \mathrm{FT}_0 \qquad \Omega \vdash 0, \mathrm{C}, \emptyset, \mathrm{PCE}_0 \overset{\text{certify}}{\Rightarrow} \mathrm{PPS}, \mathrm{PCE}}{\Gamma \vdash_{\mathrm{lbc}} \mathrm{M}, \mathrm{FTA}, \mathrm{CE}}$$

où $\mathrm{PCE}_0 = \langle \mathrm{P}_0, \langle \mathrm{FTC}_0, \emptyset \rangle \rangle, \mathrm{PCE} = \langle \mathrm{P}_0, \mathrm{CE} \rangle \ldots$

(6.2.4b)
$$\frac{\mathrm{CH} \vdash \mathrm{FTA}(0) \sqsubset \mathrm{FT}_0 \qquad \Omega \vdash 0, \mathrm{C}, \emptyset, \mathrm{PCE}_0 \overset{\text{certify}}{\Rightarrow} \mathrm{PPS}, \mathrm{PCE}}{\Gamma \vdash_{\mathrm{lbc}} \mathrm{M}, \mathrm{FTA}, \mathrm{CE}}$$

où $\mathrm{PCE}_0 = \langle \mathrm{P}_0, \langle \mathrm{FTC}_0 \sqcap \mathrm{FTA}_0, \emptyset \rangle \rangle, \mathrm{PCE} = \langle \mathrm{P}_0, \mathrm{CE} \rangle \ldots$

Vérification standard d'une séquence d'instructions

La règle simple **(4.2.8 a)**: (dernière instruction)

$$\frac{\Omega \vdash \mathrm{PP}, \mathrm{I} \overset{\mathrm{tsafe}}{\rightarrow} \mathrm{PP}', \top}{\Omega \vdash \mathrm{PP}, \mathrm{I}, \mathrm{PPS} \overset{\mathrm{tsafe}}{\Rightarrow} \mathrm{PPS}'} \qquad \text{où } \mathrm{PPS}' = \mathrm{PPS} \cup \{\mathrm{PP}\}$$

La règle composée **(4.2.8 b)**:

$$\frac{\begin{array}{l}\Omega \vdash \mathrm{PP} : \mathrm{I}_1 \overset{\mathrm{tsafe}}{\rightarrow} \mathrm{PP}', \mathrm{FT}^{\mathrm{PP}}_{\mathrm{PP}'} \\ \mathrm{CH} \vdash \mathrm{FTA}(\mathrm{PP}') \sqsubseteq \mathrm{FT}^{\mathrm{PP}}_{\mathrm{PP}'} \\ \Omega \vdash \mathrm{PP}', \mathrm{I}_2 \cdot \mathrm{CS}, \mathrm{PPS}' \overset{\mathrm{tsafe}}{\Rightarrow} \mathrm{PPS}''\end{array}}{\Omega \vdash \mathrm{PP}, \mathrm{I}_1 \cdot \mathrm{I}_2 \cdot \mathrm{CS}, \mathrm{PPS} \overset{\mathrm{tsafe}}{\Rightarrow} \mathrm{PPS}''}$$

$$\text{où} \quad \begin{array}{r}\Omega = \langle \Gamma, \_, \mathrm{FTA} \rangle \\ \Gamma = \langle \_, \mathrm{CH} \rangle \\ \mathrm{PPS}' = \mathrm{PPS} \cup \{\mathrm{PP}\}\end{array}$$

Certification d'une séquence

La règle simple (6.2.9a): (dernière instruction)

$$\frac{\Omega \vdash \mathrm{PP}, \mathrm{I}, \mathrm{PCT} \overset{\text{certify}}{\rightarrow} \mathrm{PP}', \top, \mathrm{PCT}_0}{\Omega \vdash \mathrm{PP}, \mathrm{I}, \mathrm{PPS}, \mathrm{PCE} \overset{\text{certify}}{\Rightarrow} \mathrm{PPS}', \mathrm{PCE}'}$$

où

$$\mathrm{PCE} = \langle \mathrm{P}, \mathrm{CE} \rangle$$

$$\mathrm{CE} = \langle \_, \mathrm{LS} \rangle$$

$$\mathrm{PCT} = \langle \mathrm{P}, \mathrm{LS} \rangle$$

$$\mathrm{PCT}_0 = \langle \mathrm{P} \sqcup \{\mathrm{PP} \mapsto \top\}, \mathrm{LS} \rangle \quad \textit{Condition finale}$$

$$\mathrm{PPS}' = \mathrm{PPS} \cup \{\mathrm{PP}\} \quad \textit{en commun avec} \ (4.2.8\mathrm{a.i})$$

$$\mathrm{PCE}' = \langle \mathrm{P}_0, \mathrm{CE} \rangle$$

Certification d'une séquence

$$\mathrm{CH} \vdash \mathrm{FTA}(\mathrm{PP}') = (\mathrm{FT}^{\mathrm{PP}}_{\mathrm{PP}'} \sqcap \mathrm{P}(\mathrm{PP}'))$$

$$\Omega \vdash \mathrm{PP} : \mathrm{I}_1, \mathrm{PCT} \overset{\text{certify}}{\rightarrow} \mathrm{PP}', \mathrm{FT}^{\mathrm{PP}}_{\mathrm{PP}'}, \mathrm{PCT}'$$

La règle composée (6.2.9b): $$\frac{\Omega \vdash \mathrm{PP}', \mathrm{I}_2 \cdot \mathrm{CS}, \mathrm{PPS}', \mathrm{PCE}' \overset{\text{certify}}{\Rightarrow} \mathrm{PPS}'', \mathrm{PCE}''}{\Omega \vdash \mathrm{PP}, \mathrm{I}_1 \cdot \mathrm{I}_2 \cdot \mathrm{CS}, \mathrm{PPS}, \mathrm{PCE} \overset{\text{certify}}{\Rightarrow} \mathrm{PPS}'', \mathrm{PCE}''}$$

$$\mathrm{CH} \vdash \mathrm{FTA}(\mathrm{PP}') \sqsubset (\mathrm{FT}^{\mathrm{PP}}_{\mathrm{PP}'} \sqcap \mathrm{P}(\mathrm{PP}'))$$

$$\Omega \vdash \mathrm{PP} : \mathrm{I}_1, \mathrm{PCT} \overset{\text{certify}}{\rightarrow} \mathrm{PP}', \mathrm{FT}^{\mathrm{PP}}_{\mathrm{PP}'}, \mathrm{PCT}'$$

La règle composée **(6.2.9 c)**: $$\frac{\Omega \vdash \mathrm{PP}', \mathrm{I}_2 \cdot \mathrm{CS}, \mathrm{PPS}', \mathrm{PCE}' \overset{\text{certify}}{\Rightarrow} \mathrm{PPS}'', \mathrm{PCE}''}{\Omega \vdash \mathrm{PP}, \mathrm{I}_1 \cdot \mathrm{I}_2 \cdot \mathrm{CS}, \mathrm{PPS}, \mathrm{PCE} \overset{\text{certify}}{\Rightarrow} \mathrm{PPS}'', \mathrm{PCE}''}$$

Les conditions sémantiques

$$\Omega = \langle \Gamma, \_, \mathrm{FTA} \rangle \quad \textit{en commun avec} \ (4.2.8\text{b.i})$$

$$\Gamma = \langle \_, \mathrm{CH} \rangle \quad \textit{en commun avec} \ (4.2.8\text{b.ii})$$

$$\mathrm{PPS}' = \mathrm{PPS} \cup \{\mathrm{PP}\} \quad \textit{en commun avec} \ (4.2.8\text{b.iii})$$

$$\mathrm{PCE} = \langle \mathrm{P}, \mathrm{CE} \rangle$$

$$\mathrm{P}' = \mathrm{P} \sqcup \{\mathrm{PP} \mapsto \top\}$$

$$\mathrm{PCT} = \langle \mathrm{P}', \mathrm{LS} \rangle$$

$$\mathrm{PCT}' = \langle \mathrm{P}'', \mathrm{LS}' \rangle$$

$$\mathrm{CE} = \langle \mathrm{LS}, \mathrm{FTC} \rangle$$

$$\mathrm{CE}' = \langle \mathrm{LS}', \mathrm{FTC} \rangle \ (6.2.9\text{b}) \quad \textit{ou} \quad \mathrm{CE}' = \langle \mathrm{LS}', \mathrm{FTC} \sqcap \{\mathrm{PP}' \mapsto \mathrm{FTA}(\mathrm{PP}')\} \rangle \ (6.2.9\text{c})$$

$$\mathrm{PCE}' = \langle \mathrm{P}', \mathrm{CE}' \rangle$$

$$\mathrm{PCE}'' = \langle \mathrm{P}_0, \mathrm{CE}'' \rangle$$

Lemme 1

$$\Omega \vdash \mathrm{PP} : \mathrm{I}_1 \overset{\text{tsafe}}{\rightarrow} \mathrm{PP}', \mathrm{FT}^{\mathrm{PP}}_{\mathrm{PP}'} \wedge \mathbf{Inv1}(\mathrm{PP})$$

$$\Longrightarrow \Omega \vdash \mathrm{PP} : \mathrm{I}_1, \mathrm{PCT} \overset{\text{certify}}{\rightarrow} \mathrm{PP}', \mathrm{FT}^{\mathrm{PP}}_{\mathrm{PP}'}, \mathrm{PCT}' \wedge \mathbf{Inv1}(\mathrm{PP}')$$

Invariant : $(\mathbf{Inv1}(\mathrm{PP}))$

- les "labels" LS sont définies par les règles de certification d'une instruction,
  $\mathrm{LS} = \{\mathrm{PP}'' \mid \forall \mathrm{FT}^{\mathrm{PP}'}_{\mathrm{PP}''} : (\mathrm{PP}'' \leq \mathrm{PP}' \leq \mathrm{PP}) \wedge (\mathrm{FTA}(\mathrm{PP}'') \sqsubseteq \mathrm{FT}^{\mathrm{PP}'}_{\mathrm{PP}''})\}$
- le P est définie par les règles de certification d'une instruction,
  $\mathrm{P} = \{\mathrm{PP}'' \mapsto \mathrm{FT}^{\mathrm{PP}'}_{\mathrm{PP}''} \sqcap \mathrm{P}(\mathrm{PP}'') \mid \forall \mathrm{FT}^{\mathrm{PP}'}_{\mathrm{PP}''} : (\mathrm{PP}' \leq \mathrm{PP} < \mathrm{PP}'')\}$

Démonstration. Par verification de tous les cas des règles (voir tableau). □

Certification des instructions, mise à jour.

Règle	Saut de PP	"Labels"	"Pending"	contraintes
$(*)$	—	—	—	—
$(**)$	—	**RègleExcp**	**RègleExcp**	**RègleExcp**
(6.3.8a), (6.3.9a)	$\mathrm{PP}'' \leq \mathrm{PP}$	$\mathrm{LS} \cup \{\mathrm{PP}''\}$	—	$\mathrm{FTA}(\mathrm{PP}'') \sqsubseteq \mathrm{FT}^{\mathrm{PP}''}_{\mathrm{PP}}$
(6.3.8b), (6.3.9b)	$\mathrm{PP}'' > \mathrm{PP}$	—	$\mathrm{P} \sqcap \{\mathrm{PP}'' \mapsto \mathrm{FT}^{\mathrm{PP}}_{\mathrm{PP}''}\}$	$\mathrm{FTA}(\mathrm{PP}'') \sqsubseteq \mathrm{FT}^{\mathrm{PP}''}_{\mathrm{PP}}$

RègleExcp:

Règle	Saut de PP	"Labels"	"Pending"	contraintes
$(***)$	—	—	—	—
(6.4.5a), (6.4.6a)	$\mathrm{PP}'' \leq \mathrm{PP}$	$\mathrm{LS} \cup \{\mathrm{PP}''\}$	—	$\mathrm{FTA}(\mathrm{PP}'') \sqsubseteq \mathrm{FT}^{\mathrm{PP}''}_{\mathrm{PP}}$
(6.4.5b), (6.4.6b)	$\mathrm{PP}'' > \mathrm{PP}$	—	$\mathrm{P} \sqcap \{\mathrm{PP}'' \mapsto \mathrm{FT}^{\mathrm{PP}}_{\mathrm{PP}''}\}$	$\mathrm{FTA}(\mathrm{PP}'') \sqsubseteq \mathrm{FT}^{\mathrm{PP}''}_{\mathrm{PP}}$

où (*) est (6.3.2 a), (6.3.3 a), (6.3.11 a), (6.3.11 b), ou (6.3.11 c) ; (**) est (6.3.4a), (6.3.5 a), (6.3.6 a), (6.3.7 b), (6.3.10 a), (6.3.10 b), ou (6.3.10 c) ; (***) est (6.4.6a), (6.4.6b), (6.4.6c), (6.4.6d), (6.4.6e).

Lemme 2

$$\mathrm{CH} \vdash \mathrm{FTA}(\mathrm{PP}') \sqsubseteq \mathrm{FT}^{\mathrm{PP}}_{\mathrm{PP}'} \wedge \mathbf{Inv1}(\mathrm{PP}') \implies \mathrm{CH} \vdash \mathrm{FTA}(\mathrm{PP}') \sqsubseteq (\mathrm{FT}^{\mathrm{PP}}_{\mathrm{PP}'} \sqcap \mathrm{P}(\mathrm{PP}'))$$

Démonstration. Remarques que PP', FTA, et CH sont définis par les mêmes conditions, partagés avec (bv). L'invariant **Inv1(PP')** implique que $\mathrm{P}(\mathrm{PP}')$ est bien défini comme un contrainte de typage en PP', *i.e.*, $\mathrm{CH} \vdash \mathrm{FTA}(\mathrm{PP}') \sqsubseteq \mathrm{P}(\mathrm{PP}')$. En supposant que $\mathrm{CH} \vdash \mathrm{FTA}(\mathrm{PP}') \sqsubseteq \mathrm{FT}^{\mathrm{PP}}_{\mathrm{PP}'}$, nous avons
$\mathrm{CH} \vdash \mathrm{FTA}(\mathrm{PP}') \sqsubseteq (\mathrm{FT}^{\mathrm{PP}}_{\mathrm{PP}'} \sqcap \mathrm{P}(\mathrm{PP}'))$. □

Hypothèse

$$\Omega \vdash \mathrm{PP}', \mathrm{I}_2 \cdot \mathrm{CS}, \mathrm{PPS}' \overset{\text{tsafe}}{\Rightarrow} \mathrm{PPS}'' \wedge \mathbf{Inv1}(\mathrm{PP}') \wedge \mathbf{Inv2}(\mathrm{PP})$$

$$\Longrightarrow \Omega \vdash \mathrm{PP}', \mathrm{I}_2 \cdot \mathrm{CS}, \mathrm{PPS}', \mathrm{PCE}' \overset{\text{certify}}{\Rightarrow} \mathrm{PPS}'', \mathrm{PCE}'' \wedge \mathbf{Inv2}(\mathrm{PP}')$$

Invariant: $(\mathbf{Inv2}(\mathrm{PP}))$

- Les "frame type certificates" FTC, sont définies par les règles de séquence.
  $\mathrm{FTC} = \{\mathrm{PP}' \mapsto \mathrm{FTA}(\mathrm{PP}') \mid \mathrm{PP}' \leq \mathrm{PP} \wedge \mathrm{FTA}(\mathrm{PP}') \sqsubset (\mathrm{FT}_{\mathrm{PP}'}^{\mathrm{PP}'^{-1}} \sqcap \mathrm{P}(\mathrm{PP}'))\}$
- Le P est définie par les règles de séquence. $\mathrm{P} = \{\mathrm{PP}' \mapsto \top \mid \mathrm{PP}' \leq \mathrm{PP}\}$

Démonstration

Nous montrerons, par récurrence dans la longeur de la code, que

$$\Omega \vdash \mathrm{PP}, \mathrm{I}_1 \cdot \mathrm{I}_2 \cdot \mathrm{CS}, \mathrm{PPS}, \rightarrow \mathrm{PPS}'' \wedge \mathbf{Inv1+2}(\mathrm{PP})$$

implique que

$$\Omega \vdash \mathrm{PP}, \mathrm{I}_1 \cdot \mathrm{I}_2 \cdot \mathrm{CS}, \mathrm{PPS}, \mathrm{PCE} \overset{\text{certify}}{\Rightarrow} \mathrm{PPS}'', \mathrm{PCE}'' \wedge \mathbf{Inv1+2}(\mathrm{PP}')$$

en utilisant les lemmes. □

Cadre

Problème

Conception

Formalisation

Réalisation

Conclusion

Exemples

Prototype

- Réalisé en Java: 40k de source donnant 26k en bytecode.
- Utilise "Byte Code Engineering Library" de Markus Dahm (l'interface prends 33k de source donnant 20k en bytecode).

Utilisation de la mémoire.

	Scratch	Flash	Commentaires
CE		✓	
$\mathrm{FT}_{\texttt{actuel}}$	$(\mathrm{MS}+\mathrm{ML})$		
S	$\#\mathrm{LS}\times(\mathrm{MS}+\mathrm{ML})$		peut réutiliser CE
P	$\#\mathrm{P}\times(\mathrm{MS}+\mathrm{ML})$		peut réutiliser CE

L'éxecution avec le prototype du "Lightweight"

```
METHOD 'm':
CERTIFICATE ce =({11},
  {11:<*, LE;.bot>},
  1)
VERIFICATION fta =
- <*, LE;.bot>
0: new LA;
- <LA;, LE;.bot>
7: astore 1
- <*, LE;.LA;>
8: goto +11
- <*, LE;.bot>
11: new LB:
- <B, LE;.bot>
18: astore 1
- <*, LE;.LB;>
19: aload 1
- <LB;, LE;.LB;>
20: invokevirtual methodref(LB;, methsig(dummy, []), I)
- <I, LE;.LB;>
23: ifne -12
- <*, LE;.bot>
26: return
- top
VERIFICATION OF 'm' OK.
```

Le KVM de Sun

Un cértificat élaboré permet de parcourir le code en avant.

Simulation par LBV. Si un certificat CE contient tout constraint de typage aux buts des sauts, S et P ne sont plus necessaires, étant donné que les contraintes de typage sont vérifiés immédiate avec le certificat.

Utilisation de la mémoire.

	Scratch	Flash	Commentaires
CE		✓	proportionel aux nombres de sauts
S			n'existe plus
P			n'existe plus
FT_{actuel}	✓		

Le "on-card" verificateur de Leroy

Des prè-conditions sur le code permets de le parcourir en avant:

Une *transformation du code* avant vérification, permet d'introduire des invariants de typage tel que ST soit vide à tout les but de sauts. En plus, *chaque location dans le tableau des variables loceaux est présumé d'être typé*, ce qui restreint l'utilisation, mais fait que tout comparaisons entre LT disparaissent.

Simulation par LBV. Avec les mêmes contraintes sur le code come pour le "on-card" vérificateur, le certificat CE revient à des valeurs $\langle \epsilon, \mathrm{LS} \rangle$. Les structures S et P reviennent a des valeurs $\langle \epsilon, \mathrm{LT} \rangle$, ce qui en effets peu être optimisé a une type de "frame" actuel, et quels points de programme qui sont des buts de sauts.

Cadre

Problème

Conception

Formalisation

Réalisation

Conclusion

Exemples

Conclusion

Nous avons obtenu

- une application industrielle importante: la **off-device pre-verifier**, la **in-device verifier** et le certificat **StackMap** de KVM,
- une formalisation constructive de la technique pour une sous-ensemble importante de la JVM,
- une démonstration que la technique du "lightweight" soit fiable par rapport à la vérificateur standard,
- la **synthèse** formelle de KVM, la "on-card" vérificateur Java et la technique de la vérification lightweight,
- une réalisation d'un prototype de lightweight écrit en Java, qui utilise de mémoire donnée: $\#\text{type descriptors} = (\text{MS} + \text{ML}) \times (1 + \#\text{LS} + \#\text{P})$

Travaux relatés

- Vérificateur pour KVM: Liang 1999.
- "On-card" vérificateur: Leroy 2001.
- Preuve mécanisé de LBV: Klein and Nipkow 2000.
- Autre: Barthe et al. 2000.
- "Type safety": Drossopoulou et al. 1999, Oheimb 1998, Goldberg et al. 2000, Ancona et al. 2001.
- jsr: Freund 1998, Stata et Abadi 1998.
- Initialisation: Freund Mitchell 1998.
- "Class loading": Jensen et al. 1997.
- General: Hartel et Moreau 2001, Leroy 2001.

Cadre

Problème

Conception

Formalisation

Réalisation

Conclusion

Exemples

Arrays

```
class A {}
class Aarray {
   void m() {
     A x = new A();
     A[] array = new A[10];
     Object obj;
     array[0] = x;
     ((Object[]) array)[0] = x;
     x = array[0];
   }
}
```

```
LIGHTWEIGHT BYTECODE VERIFICATION of 'Aarray':

CLASS HIERARCHY ch = { Ljava/lang/Object; <: bot
 , Ljava/lang/Throwable; <: Ljava/lang/Object;
 , Ljava/lang/Exception; <: Ljava/lang/Throwable;
 , LAarray; <: Ljava/lang/Object;
 , LA; <: Ljava/lang/Object;
 }

METHOD 'm':
CERTIFICATE ce =({},
  {},
  0)
VERIFICATION fta =
- <*, LAarray;.bot.bot.bot>
0: new LA;
- <LA;, LAarray;.bot.bot.bot>
7: astore 1
- <*, LAarray;.LA;.bot.bot>
8: iconst 0
- <I, LAarray;.LA;.bot.bot>
10: anewarray LA;
- <[LA;, LAarray;.LA;.bot.bot>
13: astore 2
- <*, LAarray;.LA;.[LA;.bot>
14: aload 2
- <[LA;, LAarray;.LA;.[LA;.bot>
15: iconst 0
- <[LA;.I, LAarray;.LA;.[LA;.bot>
16: aload 1
- <[LA;.I.LA;, LAarray;.LA;.[LA;.bot>
17: aastore
- <*, LAarray;.LA;.[LA;.bot>
18: aload 2
- <[LA;, LAarray;.LA;.[LA;.bot>
19: checkcast [Ljava/lang/Object;
- <[LA;, LAarray;.LA;.[LA;.bot>
22: iconst 0
- <[LA;.I, LAarray;.LA;.[LA;.bot>
23: aload 1
- <[LA;.I.LA;, LAarray;.LA;.[LA;.bot>
24: aastore
- <*, LAarray;.LA;.[LA;.bot>
25: aload 2
- <[LA;, LAarray;.LA;.[LA;.bot>
26: iconst 0
- <[LA;.I, LAarray;.LA;.[LA;.bot>
27: aaload
- <LA;, LAarray;.LA;.[LA;.bot>
28: astore 1
- <*, LAarray;.LA;.[LA;.bot>
29: return
- top
VERIFICATION OF 'm' OK.
```

Exceptions

```
class A { void mA() {} }
class B extends A { void mB() {} }
class C extends A {}
class X extends Exception {}
class Xx {
  void x() throws X { throw new X(); }
  void m() {
    A a = new A();
    try {
      x();
      B b = new B(); B b2 = b;
      x();
    }
    catch (X x) {
      A c = a;
      c.mA();
} } }
```

```
LIGHTWEIGHT BYTECODE VERIFICATION of 'Xx':

CLASS HIERARCHY ch = { Ljava/lang/Object; <: bot
 , Ljava/lang/Throwable; <: Ljava/lang/Object;
 , Ljava/lang/Exception; <: Ljava/lang/Throwable;
 , LA; <: Ljava/lang/Object;
 , LB; <: LA;
 , LC; <: LA;
 , LX; <: Ljava/lang/Exception;
 , LXx; <: Ljava/lang/Object;
 }

METHOD 'm':
CERTIFICATE ce =({},
  {},
  2)
VERIFICATION fta =
- <*, LXx;.bot.bot.bot>
0: new LA;
- <LA;, LXx;.bot.bot.bot>
7: astore 1
- <*, LXx;.LA;.bot.bot>
8: aload 0
- <LXx;, LXx;.LA;.bot.bot>
9: invokevirtual methodref(LXx;, methsig(x, []), V)
- <*, LXx;.LA;.bot.bot>
12: new LB;
- <LB;, LXx;.LA;.bot.bot>
19: astore 2
- <*, LXx;.LA;.LB;.bot>
20: aload 2
- <LB;, LXx;.LA;.LB;.bot>
21: astore 3
- <*, LXx;.LA;.LB;.LB;>
22: aload 0
- <LXx;, LXx;.LA;.LB;.LB;>
23: invokevirtual methodref(LXx;, methsig(x, []), V)
- <*, LXx;.LA;.LB;.LB;>
26: goto +10
- <LX;, LXx;.LA;.bot.bot>
29: astore 2
- <*, LXx;.LA;.LX;.bot>
30: aload 1
- <LA;, LXx;.LA;.LX;.bot>
31: astore 3
- <*, LXx;.LA;.LX;.LA;>
32: aload 3
- <LA;, LXx;.LA;.LX;.LA;>
33: invokevirtual methodref(LA;, methsig(mA, []), V)
- <*, LXx;.LA;.Ljava/lang/Object;.LA;>
36: return
- top
VERIFICATION OF 'm' OK.
```