Théorème de récursion et virus informatiques

Expériences

Matthieu Kaczmarek
(Guillaume Bonfante et Jean-Yves Marion)

CARTE - LORIA - INPL

LAC 2007 - Chambéry

Définition

- Un virus informatique est un programme.
- Les virus peuvent se dupliquer.
- Les virus peuvent infecter d'autre programmes.
- Les virus peuvent muter.

D'après Cohen 1986 et Filiol 2005.

Premiers modèles mathématiques

- Cohen 1986 : machines de Turing.
  - Un virus est un mot sur le ruban.
- Adleman 1988 : fonctions récursives.
  - Un virus est une fonction récursive.
- Chess et White 2000 : programmes.
- Zuo et Zhou 2004 - 2005 : polymorphisme et complexité.

Virologie informatique abstraite

Definition (ICTAC'05)

- Un virus est un programme **v**.
- **v** est un virus relativement à une fonction de propagation B.
- $B(\mathbf{v}, \mathbf{p})$ est la forme infectée du programme **p** par **v**.
- **v** et B satisfont pour tous **p** et x, $[\![\mathbf{v}]\!](\mathbf{p}, x) = [\![B(\mathbf{v}, \mathbf{p})]\!](x)$.

- Capture et étend les définitions antérieurs (Cohen, Adleman, Zuo & Zhou).

Langage WHILE$^+$

- $\mathbb{D}$ le domaine de calcul (les arbres binaires `cons`, `nil`).
- Sémantique $[\![\ ]\!] : \mathbb{D} \to (\mathbb{D} \to \mathbb{D})$.
- Pas de distinction entre programmes et données.
- Un interpreteur (fonction universelle) $[\![\mathbf{exec}]\!](\mathbf{p}, x) = [\![\mathbf{p}]\!](x)$.
- Un spécialiseur (itération) $[\![[\![\mathbf{spec}]\!](\mathbf{p}, x)]\!](y) = [\![\mathbf{p}]\!](x, y)$.

Scenario

- Un environement est un element de $\mathbb{D}$ qui représente
  - Le système de fichiers,
  - Tous les paramètres accessibles ...
- Les programmes prennent en entrée un environment.
- Si le programme termine, sa sortie est le nouvel environment.
- Exemple : copier un élément $[\![\mathbf{copy}]\!](\mathbf{p}, x) = \langle \mathbf{p}, \mathbf{p}, x \rangle$.
- Avant : $\langle \mathbf{p}, x \rangle$, après : $\langle \mathbf{p}, \mathbf{p}, x \rangle$.
- Le programme en $\texttt{WHILE}^+$.

```
copy(p,x){
  r := cons(p,x);
  return cons(p,r);
}
```

Classification des virus

- Classification selon les critères de Cohen et Filiol.
  - Duplication du virus.
  - Infection d'hôtes.
  - Mutation du virus.
  - Mutation du processus d'infection.
- Chaque classe associée à une forme du théorème de récursion.

Classification des virus

- Classification selon les critères de Cohen et Filiol.
  - Duplication du virus.
  - Mutation du virus.
- Chaque classe associée à une forme du théorème de récursion.

Les virus blueprints

- Un *virus blueprint* pour g satistait pour tous $\mathbf{p}$, x

$$\begin{cases} \mathbf{v} \text{ est un virus relativement à un } B \text{ donné} \\ \forall \mathbf{p}, x : [\![\mathbf{v}]\!](\mathbf{p}, x) = g(\mathbf{v}, \mathbf{p}, x) \quad (g \text{ décrit le comportement}) \end{cases}$$

- Un virus blueprint utiliser son propre code pour se dupliquer.

Theorem (Théorème de récursion de Kleene)

Pour toute fonction semi-calculable g, il existe un programme $\mathbf{e}$ tel que $[\![\mathbf{e}]\!](x) = g(\mathbf{e}, x)$.

- Résout le système des virus blueprints.
- Classe des virus blueprints associée au théorème de récursion.

Construction de blueprints

Corollary

Pour tout comportement g, il existe un virus blueprint.

```
dg (z,y,x){
 r := spec(y,z,y);
 return exec(z,r,x);
}
```

Soit **g** calculant g. On définit $\mathbf{v} = [\![\mathbf{spec_2}]\!](\mathbf{dg}, \mathbf{g}, \mathbf{dg})$.

$$
\begin{aligned}
[\![\mathbf{v}]\!](\mathbf{p}, x) &= [\![\mathbf{dg}]\!](\mathbf{g}, \mathbf{dg}, \mathbf{p}, x) && \text{par la propriété d'itération} \\
&= [\![\mathbf{g}]\!]([\![\mathbf{spec_2}]\!](\mathbf{dg}, \mathbf{g}, \mathbf{dg}), \mathbf{p}, x) && \text{par définition de } \mathbf{dg} \\
&= g(\mathbf{v}, \mathbf{p}, x) && \text{par définition de } \mathbf{g} \text{ et } \mathbf{v}
\end{aligned}
$$

De plus, **v** est un virus relativement à $[\![\mathbf{spec}]\!]$. □

Exemple : le scenario de ILoveYou

- Preuve constructive et uniforme : un générateur de virus.
- **ILoveYou** $= [\![\mathbf{spec_2}]\!](\mathbf{dg}, \mathbf{g}, \mathbf{dg})$.

```
g (v, send_mail, address_book, files ) {
 /* find passwords */
 passwords := exec(find, files);
 /* send them to the attacker */
 send_mail := cons(cons(''badguy@dom.com'', passwords), send_mail);
 /* send the virus to everyone in the address_book */
 y := address_book;
 while (y) {
  send_mail := cons(cons(hd(y),v), send_mail);
  y := tl(y);
 }
 /* return the resulting environement */
 return cons(send_mail, cons(address_book, files ));
}
```

Distribution de virus

Definition

- Une *distribution de virus* est un couple de programmes $(\mathbf{d_v}, \mathbf{d_B})$.
- Pour tout i, $[\![\mathbf{d_v}]\!](i)$ est un virus relativement à la fonction de propagation $[\![[\![\mathbf{d_B}]\!](i)]\!]$.

- Un générateur de virus $\mathbf{d_v}$: muter le code
- Un générateur de propagation $\mathbf{d_B}$: muter le mode d'infection.
- Proche de la notion de compilation.

Distributions blueprints

- Une distribution blueprint pour g satistait pour tous i, $\mathbf{p}$ et x

$$\begin{cases} (\mathbf{d_v}, \mathbf{d_B}) \text{ est une distribution de virus} \\ \forall i, \mathbf{p}, x : [\![[\![\mathbf{d_v}]\!](i)]\!](\mathbf{p}, x) = g(\mathbf{d_v}, i, \mathbf{p}, x) \end{cases}$$

- Chaque virus peut générer tout autre virus de la distribution.

Theorem (Théorème de récursion de retardé - Case)

Pour toute fonction semi-calculable g, il y a un programme $\mathbf{e}$ tel que $[\![[\![\mathbf{e}]\!](x)]\!](y) = g(\mathbf{e}, x, y)$.

- Résout le système des distributions blueprints.
- Preuve constructive et uniforme : générateur de distributions.

Exemple : ILoveYou polymorphe

- Un moteur de polymorphisme : **poly** satisfait
  - $[\![\mathbf{poly}]\!](\mathbf{p}, i)$ injectif en i.
  - $[\![[\![\mathbf{poly}]\!](\mathbf{p}, i)]\!] \approx [\![\mathbf{p}]\!]$.
- $[\![\mathbf{poly}]\!]$ est une fonction de padding.

```
g (d_v, i, send_mail, address_book, files) {
  passwords := exec(find, files);
  send_mail := cons(cons("badguy@dom.com", passwords), send_mail);
  next_key := cons(nil, i)
  virus := exec(d_v, next_key);
  mutation := exec(poly, virus, i);
  y := address_book;
  while (y) {
    send_mail := cons(cons(hd(y), mutation), send_mail);
    y := tl(y);
  }
  return cons(send_mail, cons(address_book, files ));
}
```

Mieux : les smiths

- Virus smiths :
  - Utilise son code pour se reproduire.
  - Utilise un code de sa fonction de propagation pour infecter d'autre programmes.
  - Associé au théorème de récursion double.
- Distribution smiths :
  - Utilise un générateur de virus pour muter son code.
  - Utilise un générateur de propagation pour muter son mode d'infection.
  - Associé au théorème de récursion double retardé.

Conclusions

- Liens entre la théorie de la récursion et la virologie.
- Recoupe les critères de Cohen et Filiol.
- Chaque classe est associée à une forme du théorème de récursion.
- Preuves constructives : générateurs de virus.
- Le théorème de récursion de Kleene : l'ossature de la virologie informatique.