INF889A
Analyse de programmes pour la sécurité logicielle

Chapitre 6 - Interprétation abstraite

Quentin Stiévenart

Hiver 2024

https://dl.acm.org/doi/pdf/10.1145/2535838.2537850

Interprétation abstraite

Définition de Wikipédia :

> *In computer science, abstract interpretation is a theory of **sound approximation** of the **semantics of computer programs**, based on **monotonic functions over ordered sets, especially lattices**. It can be viewed as a partial execution of a computer program which gains information about its semantics (e.g., control-flow, data-flow) without performing all the calculations.*

Interprétation abstraite

- C'est une théorie de la sur-approximation
- Elle peut permettre de prouver que les analyses sont correctes
- Elle peut guider le développement d'analyses

Sûreté (*soundness*)

La sûreté d'une analyse est souvent considérée comme une propriété cruciale.

On l'a définie comme :

- tous les bugs trouvés sont valides
- l'absence de faux négatifs

On peut formaliser cela plus en détails.

Sûreté en pratique

Livshits, et al. (2015). In Defense of Soundiness: A Manifesto. CACM.

En pratique, peu d'analyse sont *sound*, car cela requiert :

- de sacrifier de la précision, et parfois rendre l'analyse très imprécise
- de complexifier l'analyse et la rendre inefficace en temps

Une analyse *soundy* :

- est majoritairement *sound*
- identifie les points d'*unsoundness*

Sûreté en pratique

Language	Examples of commonly ignored features	Consequences of not modeling these features
C/C++	`setjmp/longjmp` ignored	ignores arbitrary side-effects to the program heap
	effects of pointer arithmetic	
	“manufactured” pointers	
Java/C#	Reflection	can render much of the codebase invisible for analysis
	JNI	“invisible” code may create invisible side-effects in programs
JavaScript	eval, dynamic code loading	missing execution
	data flow through the DOM	missing data flow in program

Sur-approximation sûre (*sound over-approximation*)

Sur-approximation sûre

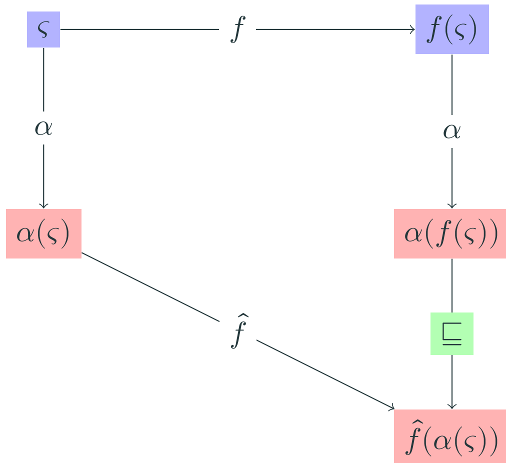
Soit :

- un domaine concret Σ
- un domaine abstrait $\hat{\Sigma}$ et sa relation d'ordre $(\sqsubseteq) \subseteq \hat{\Sigma} \times \hat{\Sigma}$
- une fonction d'abstraction $\alpha \ : \Sigma \to \hat{\Sigma}$
- une fonction sur $f : \Sigma \to \Sigma$
- une fonction sur $\hat{f} : \hat{\Sigma} \to \hat{\Sigma}$

Une fonction est une approximation **sûre** de f ssi :

$$\forall \varsigma \in \Sigma : \alpha(f(\varsigma)) \sqsubseteq \hat{f}(\alpha(\varsigma))$$

Sur-approximation sûre : graphiquement



Utilisations de la formulation

Deux approches :

- l'approche conventionnelle :
  - on définit f, α, $\hat{f}$
  - on prouve que $\hat{f}$ est une sur-approximation sûre de f
- l'approche « calculationnelle » :
  - on définit f, α
  - on énonce la condition de sûreté
  - on dérive $\hat{f}$, qui est *sound* par construction

Exemple : analyse de parité

On veut faire une interprétation de formules sur les nombres naturels pour trouver la parité d'un calcul.

On définit :

- Le domaine concret : $\Sigma = \mathbb{N} = \{0, 1, 2, \dots\}$
- Le domaine abstrait : $\hat{\Sigma} = \mathbb{P} = \{$*Even*, *Odd*$\}$
- La relation d'ordre $x \sqsubseteq y \iff x = y$
- Une fonction d'abstraction $\alpha : \Sigma \to \hat{\Sigma}$:

$$\begin{aligned} \alpha(0) &= Even \\ \alpha(n+1) &= \mathit{flip}(\alpha(n)) \\ \text{où } \mathit{flip}(Even) &= Odd \\ \mathit{flip}(Odd) &= Even \end{aligned}$$

- Une fonction f : $inc(n) = n + 1$

Analyse de parité : fonction abstraite

On définit ensuite la version abstraite : $\widehat{inc} : \mathbb{P} \to \mathbb{P}$

$$\widehat{inc}(Even) = Odd$$
$$\widehat{inc}(Odd) = Even$$

Analyse de parité : preuve classique

On peut prouver que $\widehat{inc}$ est une sur-approximation sûre, c'est-à-dire :

$$\forall n \in \mathbb{N} : \alpha(\mathit{inc}(n)) \sqsubseteq \widehat{\mathit{inc}}(\alpha(n))$$

Preuve par analyse de cas sur $\alpha(n)$

- $\alpha(n) = Even$

$$\begin{aligned}
\alpha(inc(n)) &\sqsubseteq \widehat{inc}(\alpha(n)) && \text{(déf. de sur-approximation)} \\
\alpha(inc(n)) &\sqsubseteq \widehat{inc}(Even) && \text{(hypothèse)} \\
\alpha(n+1) &\sqsubseteq Odd && \text{(déf. de } inc \text{ et } \widehat{inc}\text{)} \\
\mathit{flip}(\alpha(n)) &\sqsubseteq Odd && \text{(déf. de } \alpha\text{)} \\
\mathit{flip}(Even) &\sqsubseteq Odd && \text{(hypothèse)} \\
Odd &\sqsubseteq Odd && \text{(déf. de } \mathit{flip}\text{)}
\end{aligned}$$

- $\alpha(n) = Odd$: exercice

Analyse de parité : méthode calculationnelle

On part de la définition :

$$\alpha(inc(n)) \sqsubseteq \widehat{inc}(\alpha(n)) \qquad \text{(déf. de sur-approximation)}$$

En on inspecte au cas par cas :

- $\alpha(n) = Even$

$$\begin{aligned} \alpha(inc(n)) &\sqsubseteq \widehat{inc}(Even) && \text{(hypothèse)} \\ \alpha(n+1) &\sqsubseteq \widehat{inc}(Even) && \text{(déf. de } inc\text{)} \\ Odd &\sqsubseteq \widehat{inc}(Even) && \text{(similaire à avant)} \end{aligned}$$

  - on définit alors $\widehat{inc}(Even) \triangleq Odd$.
- $\alpha(n) = Odd$: exercice
  - on doit arriver à $\widehat{inc}(Odd) \triangleq Even$

Analyse de parité : méthode calculationnelle

On a donc *calculé* une définition sûre de $\widehat{inc}$:

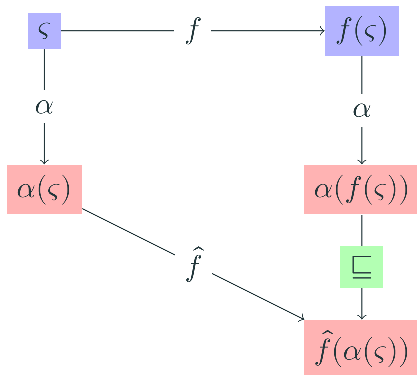
$$\widehat{inc}(Even) = Odd$$
$$\widehat{inc}(Odd) = Even$$

Utilisations de la formulation

Deux approches :

- l'approche conventionnelle :
  - on définit f, α, $\hat{f}$
  - on prouve que $\hat{f}$ est une sur-approximation sûre de f
- l'approche « calculationnelle » :
  - on définit f, α
  - on énonce la condition de sûreté
  - on dérive $\hat{f}$, qui est *sound* par construction

Sûreté et précision



La distance entre :

- l'approximation idéale $\alpha(f(\varsigma))$
- l'approximation réalisée $\widehat{f}(\alpha(\varsigma))$

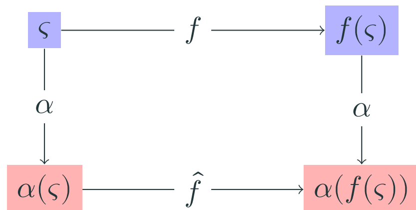
représente la *précision* que l'on a perdu

Meilleure sur-approximation possible

Meilleure sur-approximation

$\hat{f}$ est la meilleure sur-approximation (ou sur-approximation la plus précise) de f ssi

$$\forall \varsigma : \alpha(f(\varsigma)) = \hat{f}(\alpha(\varsigma))$$



Analyse de parité : précision

On a calculé la meilleure sur-approximation, car $(\sqsubseteq)$ est en fait $(=)$

$$\begin{aligned} \alpha(inc(n)) &\sqsubseteq \widehat{inc}(Even) && \text{(hypothèse)} \\ \alpha(n+1) &\sqsubseteq \widehat{inc}(Even) && \text{(déf. de } inc\text{)} \\ Odd &\sqsubseteq \widehat{inc}(Even) && \text{(similaire à avant)} \end{aligned}$$

Sémantique des langages

Sémantique des langages

On souhaite définir la sémantique d'un langage, par exemple le λ-calcul :

$$\begin{aligned} e \in \textit{Exp} ::= {} & \lambda x.e && \text{abstraction} \\ \mid {} & e_1 \ e_2 && \text{application} \end{aligned}$$

C'est un modèle Turing-complet : sa minimalité nous permet de nous concentrer sur les défis théoriques

Sémantique conventionelle du λ-calcul

$$\lambda x.e[x] \to \lambda y.e[y] \qquad \text{conversion } \alpha$$

$$(\lambda x.e_1)\ e_2 \to e_1[x := e_2] \qquad \text{réduction } \beta$$

- Avantage : c'est mathématiquement élégant
- Problème : ce n'est pas opérationnel, définir un interpréteur est compliqué à partir de cette sémantique

Felleisen, & Friedman. (1987). A Calculus for Assignments in Higher-Order Languages. POPL.

L'évaluation d'un programme est définie par 4 composantes :

- la partie **contrôle** (C) qui est l'expression à évaluer ou la valeur atteinte
- l'**environnement** (E) qui assigne à chaque variable une adresse
- le **store** (S) qui assigne à chaque adresse une valeur
- la **continuation** (K) qui représente le reste du calcul à effectuer

Machine CESK : espace d'états — Felleisen et al., POPL 1987

L'évaluation d'un programme se fait en transitionnant d'état à état

Espace d'états CESK

$$
\begin{aligned}
\sigma \in \Sigma &::= \langle c, \rho, \sigma, \kappa \rangle \\
c &::= \textbf{exp}(e) \mid \textbf{val}(v) \\
\rho &::= [x_1 \mapsto a_1, ...] \\
\sigma &::= [a_1 \mapsto v_1, ...] \\
v &::= (\lambda x.e, \rho) \\
\kappa &::= \textbf{mt} \mid \textbf{arg}(e, \rho, \kappa) \mid \textbf{fun}(v, \kappa)
\end{aligned}
$$

On peut définir une sémantique par petit pas (*small-step operational semantics*)

Fonction de transition CESK

La fonction de transition $(\longmapsto) : \Sigma \to \Sigma$ est définie par :

$$\begin{aligned}
\langle \textbf{exp}(x), \rho, \sigma, \kappa\rangle &\longmapsto \langle \textbf{val}(\sigma(\rho(x))), \rho, \sigma, \kappa\rangle \\
\langle \textbf{exp}(e_0 e_1), \rho, \sigma, \kappa\rangle &\longmapsto \langle \textbf{exp}(e_0), \rho, \sigma, \mathbf{arg}(e_1, \rho, \kappa)\rangle \\
\langle \textbf{val}(v), _, \sigma, \textbf{arg}(e, \rho', \kappa)\rangle &\longmapsto \langle \textbf{exp}(e), \rho', \sigma, \mathbf{fun}(v, \rho, \kappa)\rangle \\
\langle \textbf{val}(v), _, \sigma, \textbf{fun}((\lambda x.e, \rho'), \kappa)\rangle &\longmapsto \langle \textbf{exp}(e), \rho'[x \mapsto a], \sigma[a \mapsto v], \kappa\rangle
\end{aligned}$$

$$\text{où } a = alloc(\varsigma)$$

On commence l'évaluation du programme e par l'état :

$$\langle \mathbf{exp}(e), [], [], \mathbf{mt} \rangle$$

On applique ensuite ($\longmapsto$) jusqu'à arriver à un état final (qui ne peut plus être réduit) :

$$\textit{eval}(e) = v \iff \langle \mathbf{e}, [], [], \mathbf{mt} \rangle \longmapsto^* \langle \mathbf{val}(v), \rho, \sigma, \mathbf{mt} \rangle$$

Propriété à analyser

Une propriété P peut se représenter comme l'ensemble des éléments ayant cette propriété

Par exemple, pour une sémantique de formules mathématiques sur les entiers naturels :

- ensemble des états : $\mathcal{N}$
- propriété "est un nombre pair" : $\{0, 2, 4, ...\} \subseteq \mathcal{N}$

Propriété à analyser

Dans le cadre de l'analyse de programme, on parle souvent de propriété sur les *états atteignables*

- « l'évaluation du programme e n'atteint jamais un état ς » devient : $P = \Sigma \backslash \{\varsigma\}$
- de façon générale, on a une propriété $P \subseteq \Sigma$
  - « tous les états qui n'ont pas de *buffer overflow* »
  - « tous les états qui n'ont pas d'entrée utilisateur qui est passé à un puits »

Vérification d'une propriété

On peut définir l'ensemble des états atteignables :

$$\textit{reachable}(e) = \{\varsigma \mid \langle \mathbf{exp}(e), [], [], \mathbf{mt} \rangle \longmapsto^* \varsigma\}$$

Et la vérification d'une propriété :

$$\textit{verify}(P, e) \iff \textit{reachable}(e) \subseteq P$$

Interprétation abstraite de programmes

Problème : P et *reachable*(e) peuvent être infinis

- établir si $\varsigma \in P$ est généralement facile
- calculer *reachable*(e) est impossible

On procède donc à une sur-approximation :

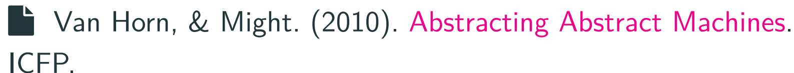
$$\textit{reachable}(e) \sqsubseteq \widehat{\textit{reachable}}(e)$$

Avec :

$$\widehat{\textit{reachable}}(e) = \{\hat{\varsigma} \mid \langle \alpha(\textbf{exp}(e), [], [], \textbf{mt}) \rangle \mathrel{\widehat{\longmapsto}}^{*} \hat{\varsigma}\}$$

Où $\alpha \ : \Sigma \rightarrow \hat{\Sigma}$ est une *fonction d'abstraction*

Modification de la sémantique Van Horn et al., ICFP 2010



Van Horn, & Might. (2010). Abstracting Abstract Machines. ICFP.

Avant tout, il faut adapter l'espace d'état pour permettre une abstraction finie :

- κ représente une liste sans borne
- on "casse" cela en mettant les continuations dans le *store*

$$
\begin{aligned}
\sigma \in \Sigma &::= \langle c, \rho, \sigma, a \rangle \\
\kappa &::= \mathbf{mt} \mid \mathbf{arg}(e, \rho, a) \mid \mathbf{fun}(v, a) \\
&\dots
\end{aligned}
$$

Modification de la sémantique Van Horn et al., ICFP 2010

Fonction de transition CESK*

La fonction de transition $(\longmapsto) : \Sigma \to \Sigma$ est définie par :

$$\langle \textbf{exp}(x), \rho, \sigma, a\rangle \longmapsto \langle \textbf{val}(\sigma(\rho(x))), \rho, \sigma, a\rangle$$

$$\langle \textbf{exp}(e_0 e_1), \rho, \sigma, a\rangle \longmapsto \langle \textbf{exp}(e_0), \rho, \sigma[b \mapsto \textbf{arg}(e_1, \rho, a), b\rangle$$

$$\text{où } b = \textit{alloc}(\varsigma)$$

$$\langle \textbf{val}(v), _, \sigma, a\rangle \longmapsto \langle \textbf{exp}(e), \rho', \sigma[b \mapsto \textbf{fun}(v, \rho, c)], b\rangle$$

$$\text{où } \sigma(a) = \textbf{arg}(e, \rho', c), b = \textit{alloc}(\varsigma)$$

$$\langle \textbf{val}(v), _, \sigma, a\rangle \longmapsto \langle \textbf{exp}(e), \rho'[x \mapsto c], \sigma[c \mapsto v], b\rangle$$

$$\text{où } \sigma(a) = \textbf{fun}((\lambda x.e, \rho'), b), c = \textit{alloc}(\varsigma)$$

On souhaite avoir un ensemble d'état **fini**. On a deux sources d'infinis :

- les continuations
- les valeurs

Les deux sont stockés dans le *store* :

- on limite le nombre d'adresse : une adresse peut désormais représenter plusieurs éléments
- on a désormais un nombre fini de valeurs
- on a désormais un nombre fini de continuations

États abstraits

$$
\begin{aligned}
\hat{\sigma} \in \hat{\Sigma} &::= \langle \hat{c}, \hat{\rho}, \hat{\sigma}, \hat{\kappa} \rangle \\
\hat{c} &::= \textbf{exp}(e) \mid \textbf{val}(\hat{v}) \\
\hat{\rho} &::= [x_1 \mapsto \hat{a}_1, ...] \\
\hat{\sigma} &::= [\hat{a}_1 \mapsto \{\hat{s}_1, \hat{s}_2, ...\}, ...] \\
\hat{s} &::= \textbf{val}(\hat{v}) \mid \textbf{kont}(\hat{\kappa}) \\
\hat{v} &::= (\lambda x.e, \hat{\rho}) \\
\hat{\kappa} &::= \textbf{mt} \mid \textbf{arg}(e, \hat{\rho}, \hat{a}) \mid \textbf{fun}(\hat{v}, \hat{a})
\end{aligned}
$$

La fonction d'abstraction $\alpha : \Sigma \to \hat{\Sigma}$ permet de passer d'un état concret à un état abstrait

Fonction d'abstraction

$$\begin{aligned}
\alpha(\langle \mathbf{exp}(e), \rho, \sigma, a \rangle) &= \langle \mathbf{exp}(e), \alpha(\rho), \alpha(\sigma), \alpha(a) \rangle \\
\alpha(\rho) &= \lambda x.\alpha(\rho(x)) \\
\alpha(\sigma) &= \lambda \hat{a}. \bigcup_{\alpha(a)=\hat{a}} \{\alpha(\sigma(a))\} \\
\alpha((\lambda x.e), \rho) &= ((\lambda x.e), \alpha(\rho)) \\
\alpha(\mathbf{mt}) &= \mathbf{mt} \\
\alpha(\mathbf{arg}(e, \rho, a)) &= \mathbf{ar}(e, \alpha(\rho), \alpha(a)) \\
\alpha(\mathbf{fun}(v, a)) &= \mathbf{fn}(\alpha(v), \alpha(a))
\end{aligned}$$

Transitions $\widehat{CESK}^*$

$$\langle \textbf{exp}(x), \hat\rho, \hat\sigma, \hat a\rangle \longmapsto \langle \textbf{val}(\hat v), \hat\rho, \hat\sigma, \hat a\rangle$$

$$\text{où } v \ni \hat\sigma(\hat\rho(x))$$

$$\langle \textbf{exp}(e_0 e_1), \hat\rho, \hat\sigma, \hat a\rangle \longmapsto \langle \textbf{exp}(e_0), \hat\rho, \hat\sigma \sqcup [\hat b \mapsto \mathbf{arg}(e_1, \hat\rho, \hat a), \hat b\rangle$$

$$\text{où } \hat b = \widehat{alloc}(\hat\varsigma)$$

$$\langle \textbf{val}(v), _, \hat\sigma, \hat a\rangle \longmapsto \langle \textbf{exp}(e), \hat\rho', \hat\sigma \sqcup [\hat b \mapsto \mathbf{fun}(\hat v, \hat\rho, \hat c)], \hat b\rangle$$

$$\text{où } \hat\sigma(\hat a) \ni \textbf{arg}(e, \hat\rho', \hat c), \hat b = \widehat{alloc}(\hat\varsigma)$$

$$\langle \textbf{val}(\hat v), _, \hat\sigma, \hat a\rangle \longmapsto \langle \textbf{exp}(e), \hat\rho'[x \mapsto \hat a], \hat\sigma \sqcup [\hat a \mapsto \hat v], \hat b\rangle$$

$$\text{où } \hat\sigma(\hat a) \ni \textbf{fun}((\lambda x.e, \hat\rho'), \hat b)$$

On a donc introduit de l'imprécision à deux endroits :

- sur les valeurs : on peut se retrouver avec plusieurs valeurs pour une variable
$$v \ni \hat{\sigma}(\hat{\rho}(x))$$
- sur les continuations : on peut se retrouver avec plusieurs continuations
$$\hat{\sigma}(\hat{a}) \ni \textbf{arg}(e, \hat{\rho}', \hat{c})$$
$$\hat{\sigma}(\hat{a}) \ni \textbf{fun}((\lambda x.e, \hat{\rho}'), \hat{b})$$

Évaluation abstraite

On a donc tous les éléments pour calculer $\widehat{\textit{reachable}}$:

$$\widehat{\textit{reachable}}(e) = \{\hat{\varsigma} \mid \langle \alpha(\textbf{exp}(e), [], [], \textbf{mt}) \rangle \mathrel{\widehat{\longmapsto}}^* \hat{\varsigma}\}$$

AAM : garanties

Décidabilité

$\hat{\varsigma} \in \widehat{\textit{reachable}}(e)$ est **décidable**

Idée de la preuve : l'espace d'état est fini.

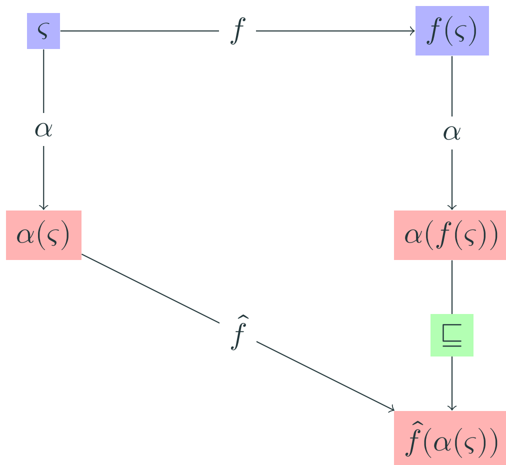
Sûreté

Si $\varsigma \longmapsto \varsigma'$ et $\alpha(\varsigma) \sqsubseteq \hat{\varsigma}$, alors il existe un état $\hat{\varsigma}'$ tel que $\hat{\varsigma} \longmapsto \hat{\varsigma}'$ et $\alpha(\varsigma') \sqsubseteq \hat{\varsigma}'$.

Idée de la preuve : par analyse de cas, en faisant l'hypothèse d'une fonction d'allocation sûre.

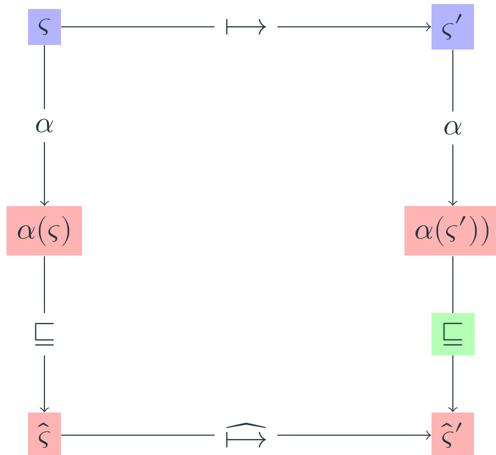
AAM : sûreté, graphiquement

Pour rappel, on avait :



AAM : sûreté, graphiquement

Si $\varsigma \longmapsto \varsigma'$ et $\alpha(\varsigma) \sqsubseteq \hat{\varsigma}$, alors il existe un état $\hat{\varsigma}'$ tel que $\hat{\varsigma} \longmapsto \hat{\varsigma}'$ et $\alpha(\varsigma') \sqsubseteq \hat{\varsigma}'$.



Interprétation abstraite : la théorie

X as abstract interpretation :

- Le typage est une forme d'interprétation abstraite : Cousot, POPL 1999
- Les analyses dataflow sont des interprétations abstraites : Schmidt, POPL 1998
- Les analyses de termination sont des interprétations abstraites : Cousot, 2012
- Les analyses de flot de contrôle sont des interprétation abstraites : Midtgaard & Jensen, ICFP 2002
- Les algorithmes de graphes sont des interprétations abstraites : Sergey et al., 2012

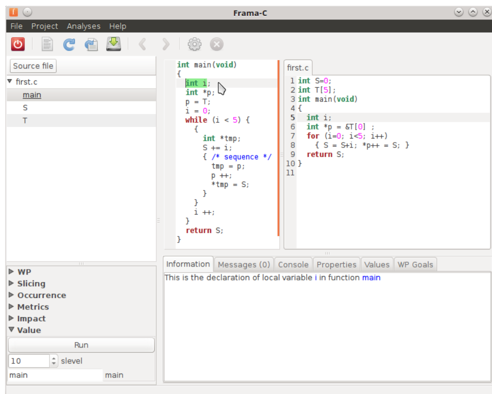
AAM en pratique

Liang, Might, & Van Horn. (2015). Anadroid: Malware Analysis of Android with User-Supplied Predicates. Electronic Notes in Theoretical Computer Science.

$$
\begin{aligned}
\hat{c} \in \widehat{State} &= \mathsf{Stmt}^* \times \widehat{FramePointer} \times \widehat{Store} \times \widehat{KontAddr} && \text{[states]}\\
\hat{\sigma} \in \widehat{Store} &= \widehat{Addr} \rightharpoonup \widehat{Val} && \text{[stores]}\\
\hat{a} \in \widehat{Addr} &= \widehat{RegAddr} + \widehat{FieldAddr} + \widehat{KontAddr} && \text{[addresses]}\\
\widehat{a_\kappa} \in \widehat{KontAddr} &\text{ is a finite set of continuation addresses} &&\\
\widehat{ra} \in \widehat{RegAddr} &= \widehat{FramePointer} \times \mathsf{Reg} &&\\
\widehat{fa} \in \widehat{FieldAddr} &= \widehat{ObjectPointer} \times \mathsf{FieldName} &&\\
\hat{\kappa} \in \widehat{Kont} &= \mathbf{fun}(\hat{fp}, \boldsymbol{s}, \widehat{a_\kappa}) + \mathbf{halt} && \text{[continuations]}\\
\hat{d} \in \widehat{Val} &= \mathcal{P}\left(\widehat{ObjectValue} + \widehat{String} + \widehat{\mathcal{Z}} + \widehat{Kont}\right) && \text{[abstract values]}\\
\widehat{ov} \in \widehat{ObjectValue} &= \widehat{ObjectPointer} \times \mathsf{ClassName} &&\\
\hat{fp} \in \widehat{FramePointer} &\text{ is a finite set of frame pointers} && \text{[frame pointers]}\\
\widehat{op} \in \widehat{ObjectPointer} &\text{ is a finite set of object pointers} && \text{[object pointers]}.
\end{aligned}
$$

Interprétation abstraite en pratique

Frama-C est une plateforme open-source d'analyse pour le C.




```
$ frama-c-gui code/framac.c
```

Interprétation abstraite en pratique

Astrée est une plateforme commerciale d'analyse pour le C et C++.

Utilisée pour vérifier du code critique : avionique, spatial, centrales nucléaires, ...

Interprétation abstraite en pratique

TODO: mailbox abstractions ECOOP

Pour aller plus loin

- Cours de Patrick Cousot au MIT (2005) : 16.399 Abstract Interpretation
  - très détaillé, revoit toutes les bases formelles
- Nielson, Nielson, & Hankin. (2015). Principles of Program Analysis. Springer.
  - chapitre 4

Articles

- Arceri, & Mastroeni. (2021). Analyzing Dynamic Code: A Sound Abstract Interpreter for Evil Eval. ACM TOPS.
- Jordan et al. (2019). Unacceptable Behavior: Robust PDF Malware Detection Using Abstract Interpretation. PLAS.
- Tiraboschi et al. (2023). Sound Symbolic Execution via Abstract Interpretation and its Application to Security. VMCAI.
- Mastroeni, & Pasqua. (2019). Statically Analyzing Information Flows: An Abstract Interpretation-Based Hyperanalysis for Non-Interference. SAC.
- Micinski et al. (2020). Abstracting Faceted Execution. CSF.
- Liang et al. (2015). AnaDroid: Malware Analysis of Android with User-Supplied Predicates. Electronic Notes in Theoretical Computer Science.