

# Analyse de flots d'information pour Caml : de la théorie à la pratique

**Vincent Simonet**

**INRIA Rocquencourt — Projet Cristal**

Vincent.Simonet@inria.fr

<http://cristal.inria.fr/~simonet/>

## Projet Cristal

Conception, implantation et établissement des fondements théoriques des langages de programmation fortement typés :

- Systèmes de types
- Programmation structurée (modules, objets, ...)
- Compilation de langages de haut niveau
- Analyses statiques

Développement du langage Caml.

► **Flots d'information**

Analyse par typage statique pour ML

Flow Caml

# Flots d'information

## Le « problème du confinement »

Les systèmes d'information exécutent simultanément **plusieurs programmes** lancés par **différents utilisateurs** qui lisent et écrivent des données dans un espace partagé.

Il est souvent nécessaire de contrôler les **flots d'information** dans ces systèmes pour préserver

- l'**intégrité** des données  
(seuls des agents autorisés peuvent modifier les données)
- la **confidentialité** (ou le secret) des données  
(seuls des agents autorisés peuvent lire les données)

[Lampson (1973)]

## Contrôle d'accès vs. contrôle de flot d'information

### Contrôle d'accès

Grâce à un système d'authentification, l'accès initial aux données est contrôlé. Aucune vérification ultérieure n'est effectuée.



Suppose une confiance envers ceux qui manipulent les données (programmes...)

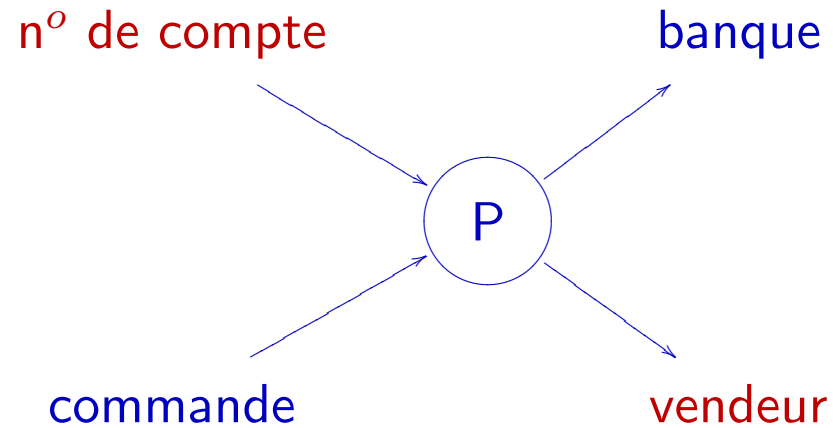
### Contrôle de flots d'information

L'ensemble des opérations effectuées par les programmes sur les données est vérifié.



Nécessite une analyse préalable des programmes

## Un exemple...



On souhaite vérifier que le **vendeur** ne peut pas recevoir d'information sur le **numéro de compte** du client.

## À quel niveau effectuer la vérification ?

**Les systèmes informatiques** sont non-déterministes et concurrents

- ils interagissent avec des périphériques extérieurs, des réseaux, des utilisateurs,
- ils sont observables par des moyens physiques (temps, consommation d'énergie, émissions...)

⇒

Le comportement d'un système informatique est difficilement analysable dans son ensemble

**Un programme** écrit dans un langage déterministe et séquentiel

- interagit par ses entrées et sorties
- a une sémantique abstraite bien définie.

⇒

Un tel programme peut être analysé formellement

## Définir la notion de flot d'information

Un programme engendre un flot d'information entre une de ses entrées  $i$  et une de ses sorties  $o$  si lorsque la valeur de  $i$  est modifiée alors celle de  $o$  est affectée.

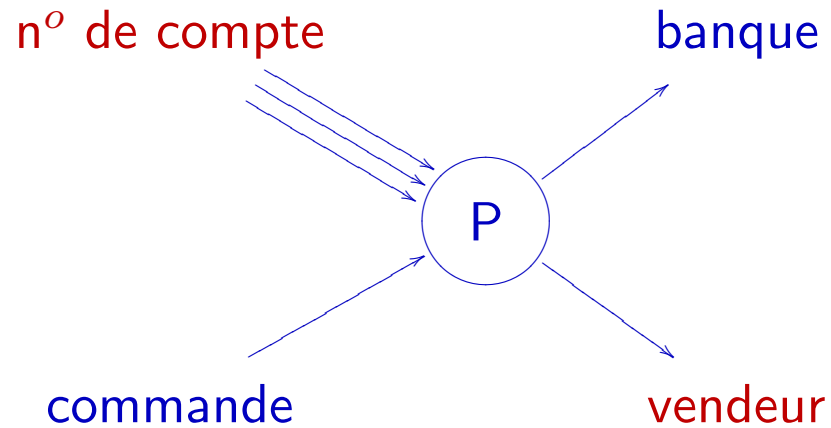
L'absence de dépendance est appelée **non-interférence**. Dans l'exemple précédent, on dit que la sortie **vendeur** ne dépend pas de l'entrée **compte** si et seulement si

$\forall \text{compte}_1, \text{compte}_2, \text{commande},$

$$\text{vendeur}(\text{commande}, \text{compte}_1) = \text{vendeur}(\text{commande}, \text{compte}_2)$$



## Définir la notion de flot d'information : exemple



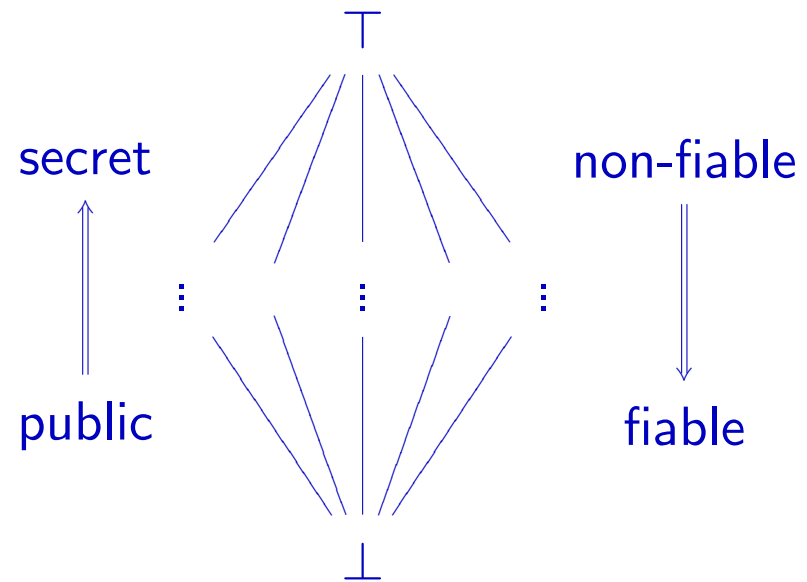
On exécute le programme avec différentes valeurs du **numéro de compte**. La valeur de sortie retournée au **vendeur** doit être inchangée.

## Définir une « politique de sécurité »

On définit généralement une politique de sécurité en utilisant un ensemble partiellement ordonné de **niveaux d'information**  $\ell \in \mathcal{L}$ .

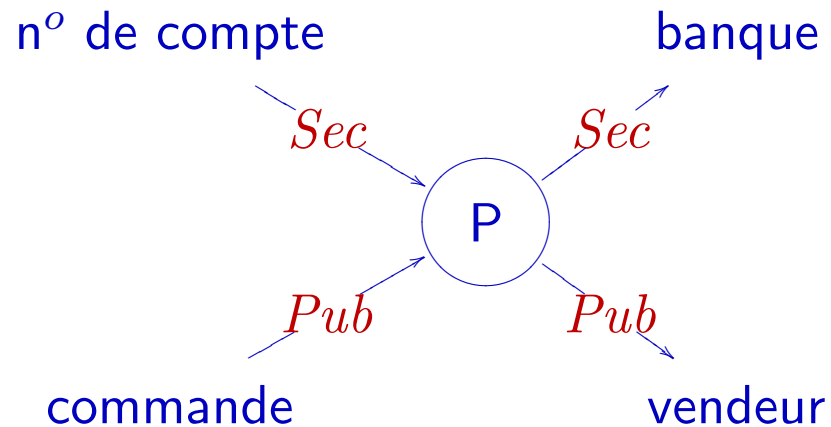
$\{Sec, Pub\}$  permet de distinguer les données **publiques** des données **secrètes** ( $Pub \leq Sec$ )

L'ensemble des parties d'un ensemble de **principaux** permet de définir, pour chaque donnée, qui peut la consulter.



[Bell & LaPadula (1973), Denning (1975)]

## Définir une « politique de sécurité » : exemple



On spécifie le comportement attendu d'un programme (vis-à-vis des flots d'information) en attribuant à chacune de ses entrées et sorties un niveau d'information.

## Analyse statique par typage

On souhaite vérifier que le comportement d'un programme est correct en analysant son code source par un système de types :

- La correction du programme analysé est vérifiée **avant son exécution**.
- Une telle analyse n'a **pas de coût à l'exécution**.
- Elle peut être effectuée de manière **automatique** (si un algorithme d'inférence est disponible).
- Les types permettent de **spécifier** le comportement des programmes.

## Un bref historique

Denning (1975–1982)

Langage impératif procédural  
Pas de preuve de correction.

Banâtre, Bryce et Le Métayer (1994)  
Volpano et Smith (1997)

Langages impératifs sans procédures.

Palsberg et Ørbæk (1995)

$\lambda$ -calcul pur.  
Pas de preuve de correction.

Heintze et Riecke (1998)  
Abadi, et al. (1999)  
Pottier et Conchon (2000)

Langage fonctionnel pur avec  
structures de données.

Myers (1999)

JAVA, avec les aspects dynamiques.  
Pas de preuve de correction. ⇒ **JIF**

Pottier et Simonet (2002)

(CA)ML, avec références et  
exceptions. ⇒ **Flow Caml**

**Flots d'information**

- ▶ **Analyse par typage statique pour ML**  
**Flow Caml**

# **Analyse par typage statique pour ML**

## Pourquoi ML ?

- ML est un langage qui possède une **sémantique** définie de manière formelle.
- ML est un langage fortement **typé**.
- ML est un langage de programmation **réaliste**, mais dont le noyau reste relativement concis.

## Core ML

### Noyau fonctionnel

<code>0, 1, 2, ...</code>	Constantes (entiers...)
<code>fun x → e</code>	Fonction
<code>e<sub>1</sub> e<sub>2</sub></code>	Application de fonction
<code>let x = e<sub>1</sub> in e<sub>2</sub></code>	Définition

### Datatypes

<code>I<sub>1</sub> e, I<sub>2</sub> e</code>	Constructeurs (unions)
<code>match e with I<sub>1</sub> x → e<sub>1</sub>   I<sub>2</sub> x → e<sub>2</sub></code>	
<code>(e<sub>1</sub>, e<sub>2</sub>)</code>	Paires
<code>fst e, snd e</code>	

### Références

<code>ref e</code>	Création d'une référence
<code>e<sub>1</sub> := e<sub>2</sub></code>	Écriture dans une référence
<code>!e</code>	Lecture d'une référence

### Exceptions

<code>raise ε e</code>	Levée d'une exception
<code>try e<sub>1</sub> with ε x → e<sub>2</sub></code>	Rattrapage d'une exception



## Types annotés

Notre analyse consiste à **annoter** les types habituels de ML avec des niveaux d'information.

Types de base	$\ell$ int, $\ell$ bool, ...
Unions	$(I_1 \text{ of } t_1 \mid I_2 \text{ of } t_2)^\ell$
Produits	$t_1 \times t_2$

Ainsi,  $\begin{cases} \text{Pub int} \\ \text{Sec int} \end{cases}$  est le type d'une expression produisant un entier  $\begin{cases} \text{public} \\ \text{secret} \end{cases}$

D'autres exemples :

$\text{Pub int} \times \text{Sec bool} \quad (I_1 \text{ of Pub int} \mid I_2 \text{ of Sec int})^{\text{Pub}}$

$\text{Sec int} \times \text{Sec bool} \quad (I_1 \text{ of Sec int} \mid I_2 \text{ of Sec int})^{\text{Sec}}$

## Sous-typage

À partir de l'ordre sur les niveaux d'information (e.g.  $Pub \leq Sec$ ), on définit une relation d'ordre entre les types.

$$\begin{array}{cc}
 \ell_1 \text{ int} \leq \ell_2 \text{ int} & t_1 \times t'_1 \leq t_2 \times t'_2 \\
 \Leftrightarrow & \Leftrightarrow \\
 \ell_1 \leq \ell_2 & t_1 \leq t'_1 \text{ et } t_2 \leq t'_2
 \end{array}$$

$$\begin{array}{c}
 (\text{I}_1 \text{ of } t_1 \mid \text{I}_2 \text{ of } t'_1)^{\ell_1} \leq (\text{I}_1 \text{ of } t_2 \mid \text{I}_2 \text{ of } t'_2)^{\ell_2} \\
 \Leftrightarrow \\
 t_1 \leq t'_1 \text{ et } t_2 \leq t'_2 \text{ et } \ell_1 \leq \ell_2
 \end{array}$$

$$\begin{array}{c}
 t'_1 \rightarrow t_1 \leq t'_2 \rightarrow t_2 \\
 \Leftrightarrow \\
 t'_2 \leq t'_1 \text{ et } t_1 \leq t_2
 \end{array}$$

## Exemple : typage de +

$$\frac{\Gamma \vdash e_1 : l_1 \text{ int} \quad \Gamma \vdash e_2 : l_2 \text{ int}}{\Gamma \vdash e_1 + e_2 : (l_1 \sqcup l_2) \text{ int}}$$

En général, on introduit une règle de **sous-typage** qui permet de simplifier l'écriture des autres règles.

$$\frac{\Gamma \vdash e_1 : l \text{ int} \quad \Gamma \vdash e_2 : l \text{ int}}{\Gamma \vdash e_1 + e_2 : l \text{ int}}$$

$$\frac{\Gamma \vdash e : t \quad t \leq t'}{\Gamma \vdash e : t'}$$

## Gardes

On introduit une forme de contraintes permettant de **marquer un type par un niveau d'information** :  $l \triangleleft t$  ( $l$  garde  $t$ ).

$$\begin{array}{ll}
 l \triangleleft l_1 \text{ int} & l \triangleleft t_1 \times t_2 \\
 \Leftrightarrow & \Leftrightarrow \\
 l \leq l_1 & l \triangleleft t_1 \text{ et } l \triangleleft t_2
 \end{array}$$

$$\begin{array}{c}
 l \triangleleft (\text{I}_1 \text{ of } t_1 \mid \text{I}_2 \text{ of } t_2)^{l_1} \\
 \Leftrightarrow \\
 l \leq l_1
 \end{array}$$

## Exemple : typage de if et match

$$\frac{\Gamma \vdash e : \ell \text{ bool} \quad \Gamma \vdash e_1 : t \quad \Gamma \vdash e_2 : t \quad \ell \triangleleft t}{\Gamma \vdash \text{if } e \text{ then } e_1 \text{ else } e_2 : t}$$

$$\frac{\Gamma \vdash e : (\text{I}_1 \text{ of } t_1 \mid \text{I}_2 \text{ of } t_2)^\ell \quad \Gamma[x \mapsto t_1] \vdash e_1 : t \quad \Gamma[x \mapsto t_2] \vdash e_2 : t \quad \ell \triangleleft t}{\Gamma \vdash \text{match } e \text{ with } \text{I}_1 x \rightarrow e_1 \mid \text{I}_2 x \rightarrow e_2 : t}$$

Dans les deux cas, la contrainte  $\ell \triangleleft t$  enregistre le fait que le résultat de l'expression globale dépend du test effectué sur  $e$ .

## Polymorphisme

On souhaite généralement utiliser la même portion de code (e.g. une fonction) avec des données de niveaux différents. Par exemple :

```
fun x → x + 1
```

peut être utilisée avec les types suivants

*Pub* int → *Pub* int      *Pub* int → *Sec* int      *Sec* int → *Sec* int

On représente cet ensemble de types par un schéma de type :

$$\forall \alpha \beta [\alpha \leq \beta]. \alpha \text{ int} \rightarrow \beta \text{ int}$$

qui peut également s'écrire

$$\forall \alpha []. \alpha \text{ int} \rightarrow \alpha \text{ int}$$

## Typage des références : Flots directs et flots indirects

### Flots directs

$x := \text{not } y$

$x := (\text{if } y \text{ then } \textit{false} \text{ else } \textit{true})$

### Flots indirects

$\text{if } y \text{ then } x := \textit{false} \text{ else } x := \textit{true}$

$x := \textit{true}; \text{ if } y \text{ then } x := \textit{false} \text{ else } ()$

## Contexte

Supposons que  $y$  contienne une donnée *secrète*.

if  $y$  then  $\underbrace{x := false}_{Sec}$  else  $\underbrace{x := true}_{Sec}$

$\underbrace{x := true}_{Pub}$ ; if  $y$  then  $\underbrace{x := false}_{Sec}$  else ()

let  $f = \text{fun } b \rightarrow \underbrace{x := b}_{?}$  in

$\underbrace{f true}_{Pub}$ ; if  $y$  then  $\underbrace{f false}_{Sec}$  else ()



## Typage des références (1/2)

$$\frac{pc, \Gamma \vdash e : \ell \text{ bool} \quad pc \sqcup \ell, \Gamma \vdash e_1 : t \quad pc \sqcup \ell, \Gamma \vdash e_2 : t \quad \ell \triangleleft t}{pc, \Gamma \vdash \text{if } e \text{ then } e_1 \text{ else } e_2 : t}$$

$$\frac{pc, \Gamma \vdash e_1 : (t, \ell) \text{ ref} \quad pc, \Gamma \vdash e_2 : t \quad \ell \sqcup pc \triangleleft t}{pc, \Gamma \vdash e_1 := e_2 : \text{unit}}$$

Exemple :  $x$  est de type  $(\ell \text{ int}, ?) \text{ ref}$

$$\underbrace{x := \text{true}}_{pc = \text{Pub} \leq \ell}; \text{ if } y \text{ then } \underbrace{x := \text{false}}_{pc = \text{Sec} \leq \ell} \text{ else } ()$$

Le système de type impose  $\text{Sec} \leq \ell$ .

## Typage des références (2/2)

$$\frac{pc', \Gamma[x \mapsto t'] \vdash e : t}{pc, \Gamma \vdash \mathbf{fun} \ x \rightarrow e : t' \xrightarrow{pc'} t}$$

$$\frac{pc, \Gamma \vdash e_1 : t' \xrightarrow{pc'} t \quad pc, \Gamma \vdash e_2 : t' \quad pc \leq pc'}{pc, \Gamma \vdash e_1 e_2 : t}$$

Exemple :  $x$  est de type  $(\ell \ \mathit{int}, ?) \ \mathit{ref}$ ,  $f$  est de type  $\mathit{Pub} \ \mathit{bool} \xrightarrow{pc'} \mathit{unit}$

let  $f = \mathbf{fun} \ b \rightarrow \underbrace{x := b}_{pc'}$  in

$\underbrace{f \ \mathit{true}}_{pc = \mathit{Pub} \leq pc'}$  ; if  $y$  then  $\underbrace{f \ \mathit{false}}_{pc = \mathit{Sec} \leq pc'}$  else  $()$

Le système de type impose  $\mathit{Sec} \leq \ell$ .

## Exceptions

Pour analyser les flots d'information engendrés par les **exceptions**, on ajoute à chaque jugement de type une **rangée** décrivant les exceptions que l'expression décrite est susceptible de lever. Par exemple :

$$pc, \Gamma \vdash e : t \quad [\text{Empty} : \text{Sec}; \text{Not\_found} : \text{Pub}]$$

Cette rangée se retrouve sur les flèches types des fonctions :

$$t' \xrightarrow{pc \quad [\text{Empty}:\text{Sec}; \text{Not\_found}:\text{Pub}]} t$$

Flots d'information

Analyse par typage statique pour ML

► Flow Caml

# Flow Caml

## Vue d'ensemble

Flow Caml est une extension du compilateur Objective Caml avec notre analyse de flots d'information.

- Supporte une grande partie du langage Caml (datatypes, valeurs mutables, exceptions, langage de modules), à l'exception des traits orientés objet.
- L'analyseur infère automatiquement les types. Il est donc inutile d'annoter les programmes.
- Il produit du code Objective Caml qui peut être compilé avec le compilateur standard.

## Définition du type list

```
type ('a, 'b) list =  
  []  
  | (::) of 'a * ('a, 'b) list  
  # 'b
```

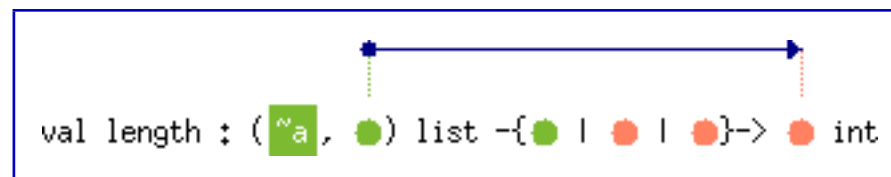
```
type (+'a:type, #'b:level) list = ...
```

## Manipulation de listes (1)

```
let rec length = function
  [] -> 0
  | _ :: t -> 1 + length t
```

```
val length : ('a, 'b) list -> 'c int
  with 'b < 'c
```

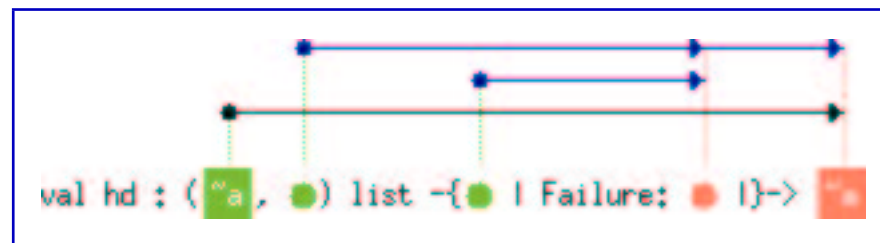
```
val length : ('a, 'b) list -> 'b int
```



## Manipulation de listes (2)

```
let rec hd = function
  [] -> failwith "List.hd"
  | h :: _ -> h
```

```
val hd : ('a, 'b) list -{'c | Failure: 'c; 'd | 'e'}-> 'a
with 'b < inter('a), 'c
```



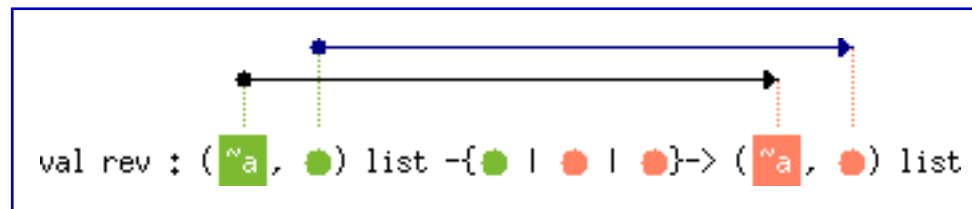


## Manipulation de listes (3)

```
let rec rev_append l1 l2 =
  match l1 with
  [] -> l2
  | a :: l -> rev_append l (a :: l2)
```

```
let rev l = rev_append l []
```

```
val rev : ('a, 'b) list -> ('a, 'b) list
```



## Définition du type queue

```
type ('a, 'b, 'c) queue =  
  { mutable in: ('a, 'b) list;  
    mutable out: ('a, 'b) list  
  }  
# 'c
```

```
type (='a:type, ='b:level, #'c:level) queue = ...
```



## Inférence de types

L'implantation de Flow Caml a nécessité l'écriture d'une bibliothèque pour l'inférence de types en présence de sous-typage, permettant :

- De **résoudre** les contraintes générées lors de l'analyse (i.e. de vérifier qu'elles ont une solution),
- De les **simplifier**, pour des raisons d'efficacité et de lisibilité des informations affichées à l'utilisateur,
- De **comparer** les schémas de types, afin de vérifier qu'une implémentation (.ml) satisfait son interface (.mli).

Cette bibliothèque pourrait être réutilisée pour d'autres analyses statiques.

## Pour en savoir plus

Une présentation de l'analyse de flots d'information, avec une preuve de correction.

François Pottier and Vincent Simonet. [Information flow inference for ML](#). In *Proceedings of the 29th ACM Symposium on Principles of Programming Languages (POPL'02)*.

La description formelle de l'algorithme d'inférence avec sous-typage structurel.

Vincent Simonet. [Type inference with structural subtyping : A faithful formalization of an efficient constraint solver](#). Submitted for publication.

<http://cristal.inria.fr/~simonet/>

[Vincent.Simonet@inria.fr](mailto:Vincent.Simonet@inria.fr)

## Limites de l'analyse

Une telle analyse de flots d'information est limitée :

- Par l'expressivité du système de types : certains programmes corrects vis-à-vis de la politique de sécurité sont rejetés par le système.
- Par la nature de la propriété de sécurité vérifiée : dans notre modèle, un programme n'est observable que par ses entrées et ses sorties.

## Quelques extensions possibles

L'utilisation de notre système nous amène à envisager des extensions.

### Au niveau de l'expressivité du langage

- traitement des **objets** (« à la ML »),
- ajout de **threads**.

### Au niveau de la flexibilité de l'analyse de flots

- traitement plus fin des **exceptions**,
- processus de **déclassification**,
- primitives **cryptographiques**,
- structures de données **hétérogènes** (types existentiels).