Inférence de flots d'information pour ML : Formalisation et implantation

Vincent Simonet

INRIA Rocquencourt — Projet Cristal

Vincent.Simonet@inria.fr
http://cristal.inria.fr/~simonet/

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

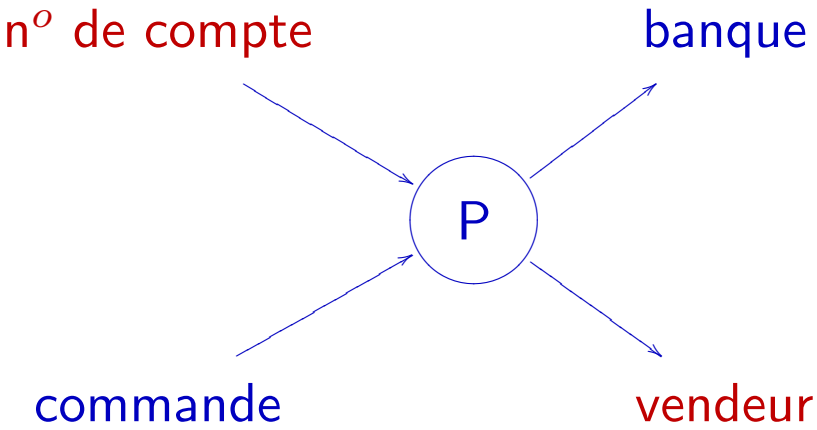
Contrôle d'accès
Grâce à un système d'authentification, l'accès initial aux données est contrôlé. Aucune vérification ultérieure n'est effectuée.

⇒ 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...)

$\Rightarrow$ 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.

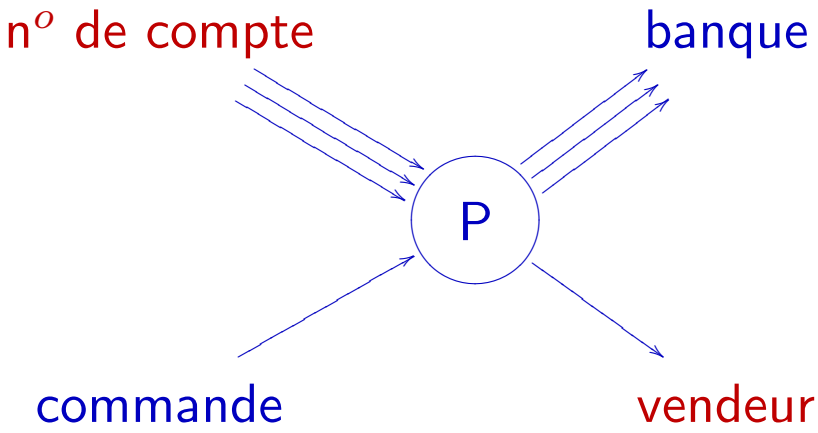
$\Rightarrow$ Un tel programme peut être analysé formellement

Définir la notion de flot d'information : la « non-interférence »

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

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

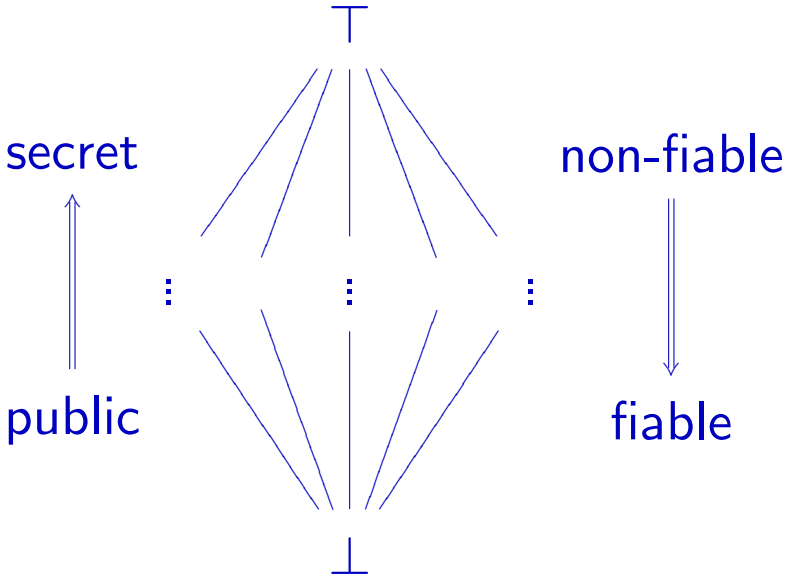


Définir une « politique de sécurité »

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

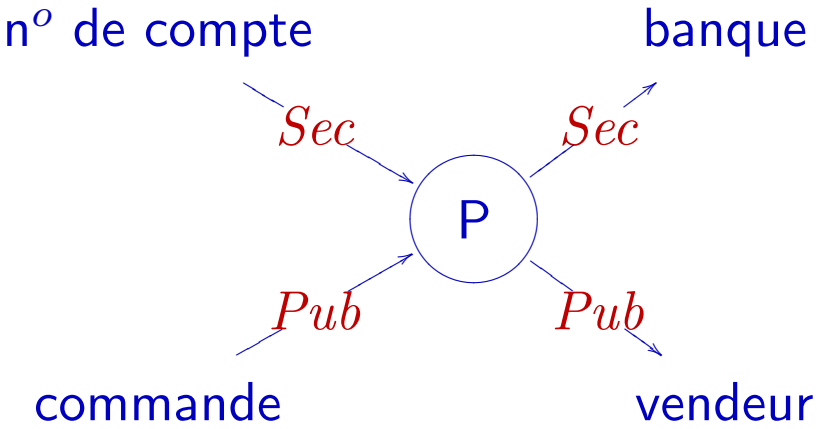
$\{Pub, Sec\}$ permet de distinguer les données publiques des données secrètes ($Pub \leq Sec$)

L'ensemble des parties d'un ensemble de principaux permet de définir qui peut accéder à chaque donnée.



[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)	λ-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. $\Rightarrow$ **JIF**
Pottier et Simonet (2002)	(CA)ML, avec références et exceptions. $\Rightarrow$ **Flow Caml**

Analyse par typage statique pour ML

Core ML

Noyau fonctionnel	$0, 1, 2, \ldots$	Constantes (entiers...)
	$\lambda x.e$	Fonction
	$e_1\, e_2$	Application de fonction
	$\texttt{let}\ x = e_1\ \texttt{in}\ e_2$	Définition
Datatypes	$\texttt{I}_1\, e,\ \texttt{I}_2\, e$	Sommes
	$\texttt{match}\ e\ \texttt{with}\ \texttt{I}_1\, x \to e_1 \mid \texttt{I}_2\, x \to e_2$	
	(e_1, e_2)	Paires
	$\texttt{fst}\ e,\ \texttt{snd}\ e$	
Références	$\texttt{ref}\ e$	Création d'une référence
	$e_1 := e_2$	Écriture dans une référence
	$!\, e$	Lecture d'une référence
Exceptions	$\texttt{raise}\ \varepsilon\, e$	Levée d'une exception
	$\texttt{try}\ e_1\ \texttt{with}\ \varepsilon\, x \to e_2$	Rattrapage d'une exception
Primitives	$f\, e$	Appel de primitive

Types annotés

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

$$
\begin{array}{rcl}
t & ::= & \texttt{int}^{\ell} \mid \texttt{bool}^{\ell} \mid \ldots \\
 & \mid & t\ \texttt{ref}^{\ell} \\
 & \mid & (t + t)^{\ell} \\
 & \mid & t \times t \\
 & \mid & (t \xrightarrow{pc\ [r]} t)^{\ell} \\
r & ::= & \{\varepsilon \mapsto \ell\}
\end{array}
$$

La relation de sous-typage est définie structurellement à partir de l'ordre sur les niveaux d'information :

$$\texttt{int}^{\oplus} \qquad \texttt{bool}^{\oplus} \qquad \odot\ \texttt{ref}^{\oplus} \qquad (\oplus + \oplus)^{\oplus} \qquad \oplus \times \oplus \qquad (\ominus \xrightarrow{\ominus\ [\oplus]} \oplus)^{\oplus}$$

Exemple : typage de $+$
(en l'absence d'effets de bord)

$$\frac{\Gamma \vdash e_1 : \mathtt{int}^{\ell_1} \qquad \Gamma \vdash e_2 : \mathtt{int}^{\ell_2}}{\Gamma \vdash e_1 + e_2 : \mathtt{int}^{(\ell_1 \sqcup \ell_2)}}$$

Grâce à la règle de sous-typage (subsumption) non dirigée par la syntaxe, on peut simplifier en :

$$\frac{\Gamma \vdash e_1 : \mathtt{int}^{\ell} \qquad \Gamma \vdash e_2 : \mathtt{int}^{\ell}}{\Gamma \vdash e_1 + e_2 : \mathtt{int}^{\ell}} \qquad \frac{\Gamma \vdash e : t \qquad t \leq t'}{\Gamma \vdash e : t'}$$

Gardes

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

$$\ell \lhd \texttt{int}^{\ell_1} \Leftrightarrow \ell \leq \ell_1 \qquad \ell \lhd t\ \texttt{ref}^{\ell_1} \Leftrightarrow \ell \leq \ell_1 \qquad \ell \lhd t_1 \times t_2 \Leftrightarrow \ell \lhd t_1 \text{ et } \ell \lhd t_2 \qquad \ell \lhd (t_1 + t_2)^{\ell_1} \Leftrightarrow \ell \leq \ell_1$$

$$\ell \lhd (t'_1 \xrightarrow{pc\ [r]} t_1)^{\ell_1} \Leftrightarrow \ell \leq \ell_1$$

Exemple : typage de `if` et `match`
(en l'absence d'effets de bord)

$$\frac{\Gamma \vdash e : \texttt{bool}^{\ell} \qquad \Gamma \vdash e_1 : t \qquad \Gamma \vdash e_2 : t \qquad \ell \lhd t}{\Gamma \vdash \texttt{if}\ e\ \texttt{then}\ e_1\ \texttt{else}\ e_2 : t}$$

$$\frac{\Gamma \vdash e : (t_1 + t_2)^{\ell} \qquad \Gamma[x \mapsto t_1] \vdash e_1 : t \qquad \Gamma[x \mapsto t_2] \vdash e_2 : t \qquad \ell \lhd t}{\Gamma \vdash \texttt{match}\ e\ \texttt{with}\ \texttt{I}_1\, x \to e_1 \mid \texttt{I}_2\, x \to e_2 : t}$$

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

Typage des références : Flots directs et flots indirects

Flots directs

$$x := \texttt{not}\ y$$

$$x := (\texttt{if}\ y\ \texttt{then}\ \mathit{false}\ \texttt{else}\ \mathit{true})$$

Flots indirects

$$\texttt{if}\ y\ \texttt{then}\ x := \mathit{false}\ \texttt{else}\ x := \mathit{true}$$

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

Contexte

Supposons que y contienne une donnée secrète.

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

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

$$\texttt{let } f = \lambda b.\underbrace{x := b}_{?} \texttt{ in}$$
$$\underbrace{f\ \mathit{true}}_{\mathit{Pub}}\texttt{; if } y \texttt{ then } \underbrace{f\ \mathit{false}}_{\mathit{Sec}} \texttt{ else } ()$$

Typage des références (1/2)
(en l'absence d'exceptions)

$$\frac{pc, \Gamma \vdash e : \texttt{bool}^{\ell} \qquad pc \sqcup \ell, \Gamma \vdash e_1 : t \qquad pc \sqcup \ell, \Gamma \vdash e_2 : t \qquad \ell \lhd t}{pc, \Gamma \vdash \texttt{if } e \texttt{ then } e_1 \texttt{ else } e_2 : t}$$

$$\frac{pc, \Gamma \vdash e_1 : t\ \texttt{ref}^{\ell} \qquad pc, \Gamma \vdash e_2 : t \qquad \ell \sqcup pc \lhd t}{pc, \Gamma \vdash e_1 := e_2 : \texttt{unit}}$$

Exemple : x est de type $\texttt{int}^{\ell}\ \texttt{ref}^{?}$

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

Le système de type impose $Sec \leq \ell$.

Typage des références (2/2)
(en l'absence d'exceptions)

$$\frac{pc', \Gamma[x \mapsto t'] \vdash e : t}{pc, \Gamma \vdash \lambda x.e : (t' \xrightarrow{pc'} t)^{\ell}}$$

$$\frac{pc, \Gamma \vdash e_1 : (t' \xrightarrow{pc'} t)^{\ell} \qquad pc, \Gamma \vdash e_2 : t' \qquad \ell \sqcup pc \leq pc' \qquad \ell \lhd t}{pc, \Gamma \vdash e_1\, e_2 : t}$$

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

$$\texttt{let } f = \lambda b.\underbrace{x := b}_{pc' \leq \ell} \texttt{ in}$$

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

Le système de type impose $Sec \leq pc' \leq \ell$.

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.

$$pc, \Gamma \vdash e : t \quad [r]$$

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

Cette rangée se retrouve sur les flèches des types de fonctions :

$$(t' \xrightarrow{pc\ [r]} t)^{\ell}$$

Typage des Exceptions

Levée d'exception

$$\frac{\Gamma \vdash v : \mathit{typexn}(\varepsilon)}{\mathit{pc}, \Gamma \vdash \texttt{raise}\ \varepsilon\, v : t \quad [\varepsilon : \mathit{pc}; \ldots]}$$

Séquencement

$$\frac{\mathit{pc}, \Gamma \vdash e_1 : t_1 \quad [r_1] \qquad \mathit{pc} \sqcup (\sqcup r_1), \Gamma \vdash e_2 : t_2 \quad [r_2]}{\mathit{pc}, \Gamma \vdash e_1; e_2 : t_2 \quad [r_1 \sqcup r_2]}$$

Rattrapage d'exception

$$\frac{\begin{array}{c}\mathit{pc}, \Gamma \vdash e_1 : t \quad [\varepsilon : \ell; r] \\ \mathit{pc} \sqcup \ell, \Gamma[x \mapsto \mathit{typexn}(\varepsilon)] \vdash e_2 : t \quad [\varepsilon : \ell'; r] \qquad \ell \lhd t\end{array}}{\mathit{pc}, \Gamma \vdash \texttt{try}\ e_1\ \texttt{with}\ \varepsilon\, x \to e_2 : t \quad [\varepsilon : \ell'; r]}$$

Preuve de non-interférence

Vue d'ensemble

1. On définit une extension particulière du langage qui permet de raisonner sur le partage entre les exécutions de deux programmes,

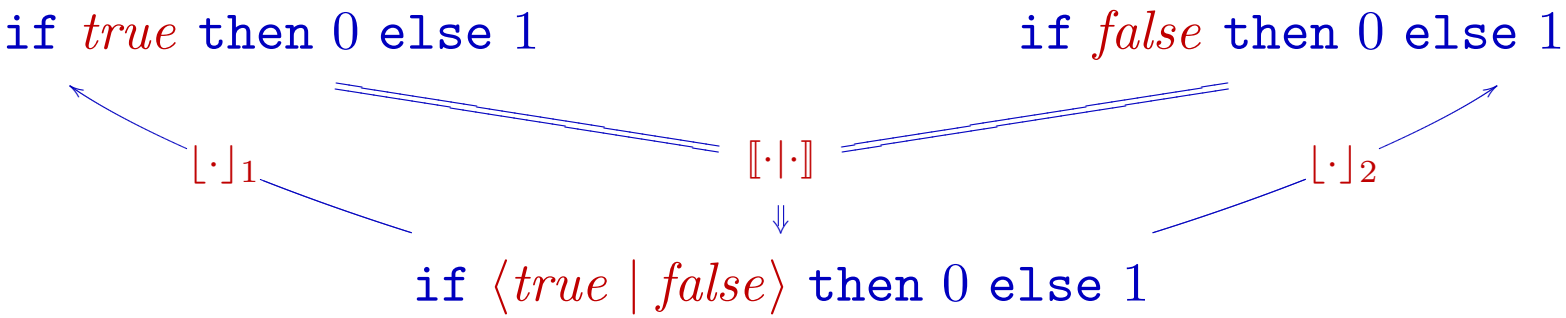
2. On prouve que le système de types pour ce langage étendu vérifie subject-reduction,

3. On en déduit la propriété de non-interférence pour le language initial.

ML avec partage : ML^2

$$e ::= \ldots \mid \langle e \mid e \rangle$$

ML^2 permet de raisonner sur deux expressions et de prouver qu'elles partagent certains sous-termes tout au long d'une réduction.



$$\langle \texttt{if } \mathit{true} \texttt{ then } 0 \texttt{ else } 1 \mid \texttt{if } \mathit{false} \texttt{ then } 0 \texttt{ else } 1 \rangle$$

Réduction dans ML^2

Les règles de réduction pour ML^2 sont dérivées de celles de ML. Quand des crochets $\langle \cdot \mid \cdot \rangle$ bloquent la réduction, ils doivent être « remontés ».

$$(\lambda x.e)\, v \to e[x \Leftarrow v] \qquad (\beta)$$

$$\langle v_1 \mid v_2 \rangle\, v \to \langle v_1 \lfloor v \rfloor_1 \mid v_2 \lfloor v \rfloor_2 \rangle \qquad (\text{lift-}\beta)$$

Exemples

$$\begin{aligned} \langle \lambda x.x \mid \lambda x.x+1 \rangle\, 3 &\to \langle (\lambda x.x)\, 3 \mid (\lambda x.x+1)\, 3 \rangle \\ &\to \langle 3 \mid (\lambda x.x+1)\, 3 \rangle \to \langle 3 \mid 4 \rangle \\ \langle \lambda x.x \mid \lambda x.x+1 \rangle\, \langle 3 \mid 2 \rangle &\to \langle (\lambda x.x)\, 3 \mid (\lambda x.x+1)\, 2 \rangle \\ &\to \langle 3 \mid (\lambda x.x+1)\, 2 \rangle \to \langle 3 \mid 3 \rangle \end{aligned}$$

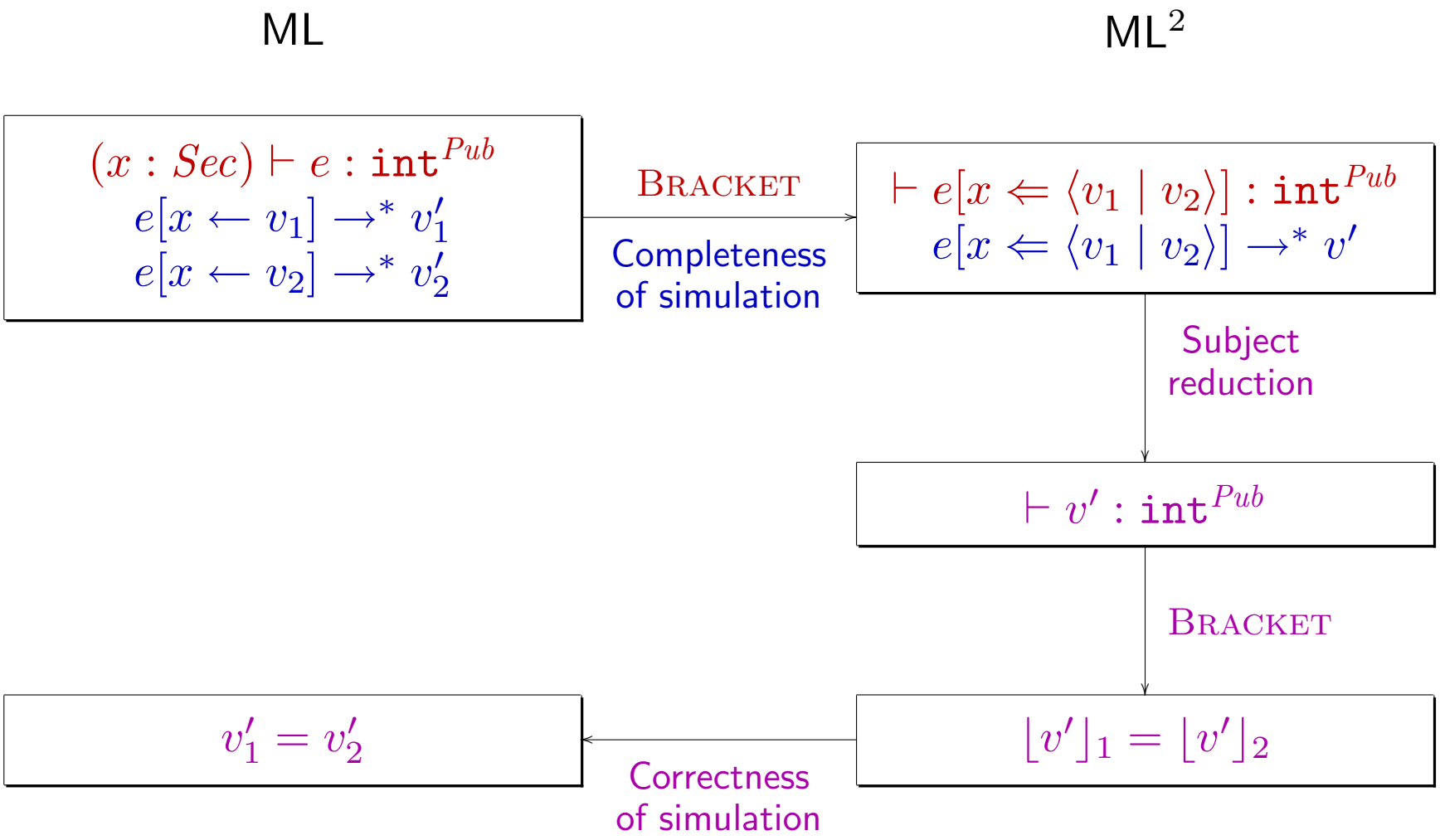
Typage de ML^2

$$\text{Bracket}\ \frac{Sec, \Gamma \vdash e_1 : t \quad [r] \qquad Sec, \Gamma \vdash e_2 : t \quad [r] \qquad Sec \lhd t}{Pub, \Gamma \vdash \langle e_1 \mid e_2 \rangle : t \quad [r]}$$

En particulier :

- Une valeur de type $\texttt{int}^{Sec}$ peut être un entier k ou un crochet d'entiers $\langle k_1 \mid k_2 \rangle$.
- Une valeur de type $\texttt{int}^{Pub}$ ne peut être qu'un entier k.

Principe de la preuve



Quelques techniques employées

Notre preuve combine plusieurs techniques orthogonales :

- Les sémantiques sont non typées. De ce fait, la preuve de simulation entre ML et ML^2 est aussi non typée.
- Le polymorphisme est traité par une approche semi-syntaxique. Ainsi, il n'intervient qu'indirectement dans la preuve.
- On introduit une ségrégation entre expressions et valeurs. Cela permet une formulation plus légère du système de types et permet de rester indépendant de la stratégie d'évaluation.
- L'invariant de la preuve est direcement codé dans les règles de typage.

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.

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.

```
type list  length  rev  hd    type queue  push  pop
                        next
```

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
```

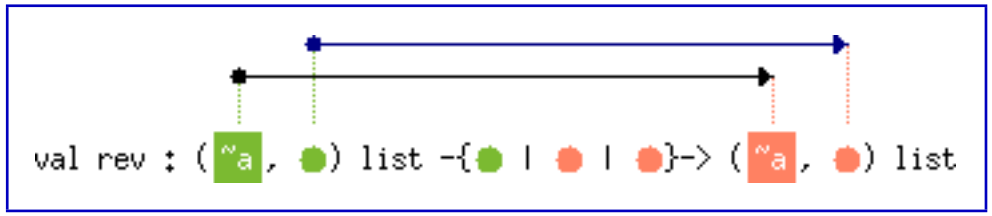
```
val hd : ('a, 'b) list -{'c | Failure: 'c; 'd | 'e}-> 'a
         with 'b < inter('a), 'c
```

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
```



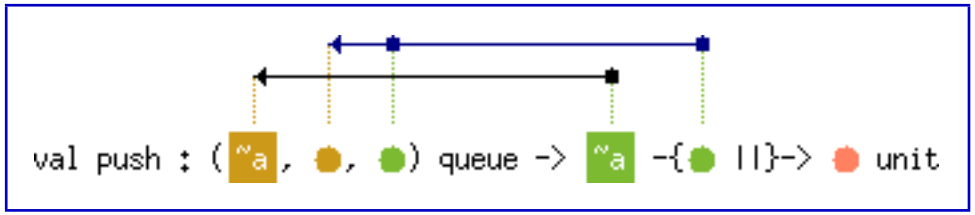
Définition du type queue

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

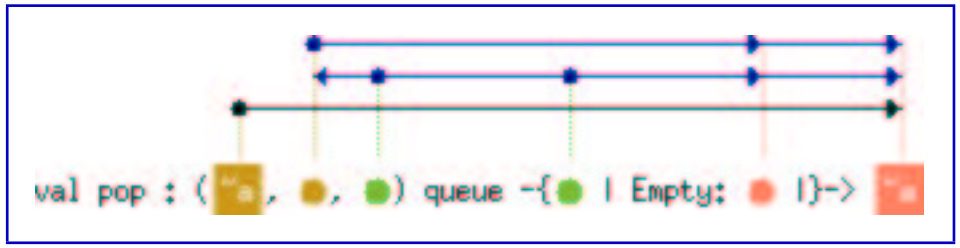
```
type (='a:type, ='b:level, #'c:level) queue = ...
```

Manipulation de queues

```
val push : ('a, 'b, 'b) queue -> 'a -{'b ||}-> _ unit
```




```
val pop : ('a, 'b, 'c) queue -{'c | Empty: 'b |}-> 'd
          with 'a < 'd
          and  'b, 'c < inter('d)
          and  'c < 'b
```



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).