Moniteur hybride de flux d'information pour un langage supportant des pointeurs

Mounir Assaf*
Julien Signoles*
Frédéric Tronel†
Éric Totel†

Résumé : Les nouvelles approches combinant contrôle dynamique et statique de flux d'information sont prometteuses puisqu'elles permettent une approche permissive tout en garantissant la correction de l'analyse réalisée vis-à-vis de la non-interférence. Dans ce papier, nous présentons une approche hybride de suivi de flux d'information pour un langage gérant des pointeurs. Nous formalisons la sémantique d'un moniteur sensible aux flux de données qui combine analyse statique et dynamique. Nous prouvons ensuite la correction de notre moniteur vis-à-vis de la non-interférence.

1 Introduction

La sécurité de l'information est traditionnellement mise en œuvre grâce à des mécanismes de contrôle d'accès. Ces mécanismes mettent en œuvre une politique de sécurité au niveau OS à un niveau gros-grain. Par contre, les mécanismes de contrôle de flux d'information (IFC) au niveau applicatif offrent une granularité plus fine pour la mise en œuvre de politiques de flux d'information précises.

Denning et Denning [1] ont initié les premiers travaux en IFC. Ils proposent une analyse statique permettant de vérifier qu'un programme propage l'information de manière correcte vis-à-vis d'une politique de flux d'information. Par exemple, une simple politique de flux d'information consisterait à interdire la fuite de variables secrètes vers des variables publiques. Goguen et Meseguer [2] généralisent cette notion en propriété de non-interférence. Depuis, cette propriété de non-interférence – la non-interférence insensible à la terminaison (TINI) plus précisément – est largement utilisée en contrôle de flux d'information en tant que propriété de sécurité [3, 4, 5, 6, 7, 8, 9, 10].

Volpano et al. [3] formalisent l'analyse de Denning et Denning en un système de types pour un langage impératif simple. Ces travaux constituent la première preuve de correction établissant qu'un programme typable respecte TINI. Cependant, le système de types proposé par Volpano et al. est insensible aux flux de données. En effet, le programme `public = secret; public = 0;` n'est pas typable dans le système de Volpano, bien qu'il ne puisse pas causer de fuites d'information car la variable publique est réinitialisée à zéro.

Hunt and Sands [4] étendent le système de types de Volpano et al. en y intégrant la sensibilité aux flux de données. Ils permettent ainsi aux labels de sécurité de refléter le

∗. CEA, LIST, Laboratoire de Sûreté des Logiciels, PC 174, 91191 Gif-sur-Yvette France, firstname.lastname@cea.fr
†. Supelec, CIDre, Rennes France, firstname.lastname@supelec.fr

niveau de sécurité de leur variable avec précision. Ils prouvent aussi la correction de leur système de types vis-à-vis de TINI. L'introduction de la sensibilité aux flux de données permet ainsi plus de permissivité, c'est-à-dire autorise un ensemble plus large de programmes sécurisés, tout en garantissant la correction de l'analyse.

L'analyse dynamique de flux d'information permet elle aussi une analyse plus permissive [8, 5] puisqu'elle accepte un plus large ensemble d'exécutions. En effet, contrairement à l'analyse statique qui met en oeuvre TINI pour l'ensemble des exécutions possibles d'un programme, l'analyse dynamique garantit TINI pour un chemin d'exécution donné tout en ignorant les chemins non exécutés. Cependant, la combinaison d'une analyse sensible aux flux de données et d'une analyse dynamique en vue de permettre plus de permissivité peut s'avérer incorrecte. En effet, Russo et Sabelfeld [8] ont prouvé que la sensibilité aux flux de données dans un moniteur de flux d'information purement dynamique introduit des canaux cachés pouvant causer des fuites d'information. L'idée principale derrière ce résultat est qu'un moniteur purement dynamique ignore les chemins non exécutés, et donc, par la même occasion, les flux d'information que ces chemins génèrent. En conséquence, un moniteur dynamique sensible aux flux de données doit s'appuyer sur une analyse statique préalable afin de garantir un contrôle de flux d'information correct vis-à-vis de TINI.

Nous formalisons dans ce papier un moniteur de flux d'information sensible aux flux de données. Ce moniteur, permissif et correct vis-à-vis de TINI, est hybride puisqu'il combine analyse dynamique et analyse statique afin de garantir sa correction. Notre moniteur supporte un langage impératif simple comprenant des pointeurs et de l'*aliasing*. Nous nous inspirons pour cela de la sémantique de Clight [11], développée dans le cadre du compilateur certifié CompCert [12]. Nous étendons en particulier nos travaux [13] en introduisant des instructions d'affichage/sortie et une notion de trace afin de permettre aux attaquants d'accéder aux mémoires intermédiaires lors de l'exécution d'un programme. Nous prouvons aussi la correction de notre moniteur vis-à-vis de TINI.

La section 2 introduit le contexte de nos travaux et plus particulièrement le contrôle de flux d'information. La section 3 présente la sémantique de notre moniteur hybride de flux d'information supportant un langage impératif simple avec pointeurs et aliasing. Nous discutons les travaux connexes dans la section 4 puis les travaux futurs dans la section 5.

2 Flux d'information

Notre modèle suppose que l'attaquant connaît le code source du programme, qu'il peut modifier les entrées publiques et lire les sorties publiques. Un programme est dit non-interférent si deux exécutions à partir de mémoires initiales qui ne diffèrent que par les entrées secrètes, fournissent les mêmes sorties publiques si ces exécutions terminent.

Généralement, des flux d'information explicites et implicites [1] sont pris en compte pour la vérification de la non-interférence. Les flux d'information explicites résultent d'un transfert d'information direct par le biais des affectations. Par exemple, l'affectation de y à x génère un flux d'information explicite de y vers x. Par contre, les flux d'information implicites résultent d'un transfert d'information conditionné par une garde. Par exemple, le programme `if (secret) x = 1 else skip` génère un flux d'information implicite de la variable *secret* à la variable x, quelle que soit la branche exécutée. En effet, même si l'affectation à x n'est pas exécutée, un attaquant pourrait déduire que *secret* est évalué à $faux$ si x est différent de 1. Nous supposerons de manière conservatoire que les affectations

au sein de conditionnelles génèrent toujours des flux d'informations implicites, afin de garantir une approximation correcte de la propriété de non-interférence.

Les pointeurs génèrent d'autres types de flux d'information. Considérons par exemple le programme `if (secret) { x = &a } else { x = &b } print *x`. Un attaquant, ayant une connaissance préalable des valeurs de a et b, pourrait déduire de l'information concernant l'évaluation de *secret* à chaque fois que $*x$ est affichée sur un canal publique : il y a donc un flux d'information de *secret* vers $*x$. En fait, ce sont deux types de flux d'information qui sont impliqués dans cet exemple. Le premier est un flux implicite de *secret* vers x. Le second est un flux d'information de x à $*x$ dû à l'*aliasing* et au déréférencement du pointeur. Ceci explique, par transitivité, le flux d'information de *secret* à $*x$.

De même, le programme `if (secret) { x = &a } else { x = &b } *x = 1` illustre la génération de flux d'information dus aux pointeurs de la variable *secret* aux variables a et b. Un attaquant ayant accès aux variables a et b après l'affectation $*x = 1$ pourrait déduire de l'information concernant l'évaluation de *secret*. Même dans le cas où a (resp. b) n'est pas affectée par l'instruction $*x = 1$, un flux d'information de *secret* à a (resp. b) est toujours produit. En effet, ces flux d'information dus aux pointeurs concernent toutes les variables correspondant à des zones mémoires qui auraient pu être écrites par l'instruction $*x = 1$ (les variables a et b dans cet exemple).

Lors de la mise en œuvre de TINI, les canaux cachés tels que la divergence des programmes ou les canaux temporels sont généralement ignorés. Par exemple, le programme *while* (*secret*) *skip* peut effectivement causer une fuite d'information concernant la variable *secret*. Cependant, cette fuite d'information est acceptable puisque, même en présence de sorties, Askarov et al. [6] ont en effet prouvé qu'un attaquant ne peut trouver le secret de manière fiable en un temps polynômial par rapport à la taille du secret.

3 Sémantique d'un moniteur de flux d'information

Langage supporté Nous présentons dans cette section un moniteur de flux d'information formalisé pour un langage avec pointeurs et aliasing. La figure 1 présente la syntaxe abstraite du langage impératif que nous considérons. Ce langage gère des pointeurs ($ptr(\tau)$) ainsi que des types de base comme des entiers (κ). Il est à noter que ce langage prend en compte l'*aliasing* mais pas l'arithmétique de pointeurs. La sémantique à grand-pas (aussi appelée sémantique naturelle [14]) de ce langage est inspirée de la sémantique du Clight [11], langage formalisé dans le cadre du compilateur certifié CompCert [12].

Une version simplifiée de la sémantique à grand pas de Clight considère un environnement E et une mémoire M. L'environnement $E : Var \rightharpoonup Loc$ associe à chaque variable une location statiquement allouée. La mémoire $M : Loc \rightharpoonup \mathbb{V}$ associe à chaque location une valeur de type τ. L'évaluation d'une instruction c dans un environnement E et une mémoire M, notée $E \vdash c, M \overset{t}{\Rightarrow} M'$, fournit une nouvelle mémoire M' ainsi qu'une trace t. Cette trace t correspond à l'observation d'une séquence de valeurs v de type τ suite à l'exécution d'instructions de sortie ou à l'observation d'une trace ϵ vide. Les expressions peuvent être évaluées en tant que valeur gauche ou droite selon la position à laquelle elles apparaissent. Toutes les expressions peuvent être évaluées en tant que valeur droite alors que seules les expressions de la forme id ou $*a$ peuvent être évaluées en tant que valeur gauche. La figure 2 illustre l'évaluation $(E \vdash a_1, M \Leftarrow l)$ de l'expression a_1 en valeur gauche, ainsi que l'évaluation $(E \vdash a_2, M \Rightarrow v)$ de l'expression a_2 en valeur droite. L'évaluation de a_1 en

$$
\begin{array}{rl}
\textbf{Types :} & \tau ::= \kappa \mid ptr(\tau) \\
\textbf{Expressions :} & a ::= n \mid id \mid uop\ a \mid a_1\ bop\ a_2 \\
 & \quad \mid *a \mid \&a \\
\textbf{Instructions :} & c ::= skip \mid a_1 = a_2 \mid c_1; c_2 \mid \text{output}_s\ a \\
 & \quad \mid if\ (a)\ c_1\ else\ c_2 \mid while\ (a)\ c \\
\textbf{Déclarations :} & dcl ::= (\tau\ id)^* \\
\textbf{Programmes :} & P ::= dcl;\ c
\end{array}
$$

Figure 1: Syntaxe abstraite du langage.

$$
(Assign)\ \frac{E \vdash a_1, M \Leftarrow l \qquad E \vdash a_2, M \Rightarrow v \qquad M' = M[l \mapsto v]}{E \vdash a_1 = a_2, M \Rightarrow M'}
$$

Figure 2: Sémantique Clight des affectations.

valeur gauche fournit la location l à laquelle a_1 est stockée, alors que l'évaluation de a_2 en valeur droite fournit la valeur v de a_2. Enfin, la règle d'affectation associe la valeur de v à la location l dans la nouvelle mémoire M'. Afin d'étendre la sémantique du Clight avec la sémantique du moniteur de flux d'information, nous considérons un treillis de label de sécurité $\mathbb{S} = (SC, \sqsubseteq)$ dont *public* est la borne inférieure. Nous notons $\sqcup$ l'opérateur d'union sur ce treillis. Nous considérons aussi une nouvelle mémoire $\Gamma : Loc \rightharpoonup \mathbb{S}$, qui associe à chaque location un label de sécurité. Informellement, la mémoire Γ reflète le niveau de sécurité du contenu de chaque location. L'affectation $x = y + z$ génère par exemple un flux d'information des variables y et z vers la variable x : Γ associe donc à la location $E(x)$ l'union $\Gamma(E(y)) \sqcup \Gamma(E(z))$ des labels associés aux locations de y et z.

Expressions Nous étendons les règles d'évaluation des expressions en valeur gauche et droite de Clight pour prendre en compte la propagation des labels de sécurité, comme illustré par la figure 3 : l'évaluation des expressions fournit une valeur $v \in \mathbb{V}$ ainsi qu'un label de sécurité $s \in \mathbb{S}$.

Si la paire (l, s_l) est le résultat de l'évaluation gauche d'une expression a, alors s_l capture les flux d'information dus aux éventuels déréférencements dans a et $s_r = \Gamma(l)$ capture les flux d'information explicites générés par la lecture de la valeur $M(l)$ associée à l'expression a. L'évaluation droite d'une expression a génère donc une valeur $v = M(l)$ et un label de sécurité $s = s_l \sqcup s_r$ prenant en compte à la fois les flux explicites et les flux dus aux pointeurs grâce à l'opérateur d'union (règle RV). Notons que la sémantique originale des expressions Clight peut être réobtenue en ignorant les opérations portant sur les labels de sécurité dans la figure 3. Nous associons à la valeur gauche de $*a$ (règle LV_{MEM}) le label de sécurité de la valeur droite de a afin de prendre en compte les flux d'information dûs aux pointeurs de a vers $*a$. Le label de sécurité *public* est associé aux valeurs droites des constantes puisque les attaquants sont supposés connaître le code source

$$LV_{ID}\ \frac{E(id) = l}{E \vdash id, M, \Gamma \Leftarrow l, public} \qquad LV_{MEM}\ \frac{E \vdash a, M, \Gamma \Rightarrow ptr(l), s}{E \vdash *a, M, \Gamma \Leftarrow l, s}$$

$$RV_{CONST}\ E \vdash n, M, \Gamma \Rightarrow n, public \qquad RV\ \frac{E \vdash a, M, \Gamma \Leftarrow l, s_l \quad M(l) = v \quad s_r = \Gamma(l) \quad s = s_l \sqcup s_r}{E \vdash a, M, \Gamma \Rightarrow v, s}$$

$$RV_{REF}\ \frac{E \vdash a, M, \Gamma \Leftarrow l, s}{E \vdash \&a, M, \Gamma \Rightarrow ptr(l), s} \qquad RV_{UOP}\ \frac{E \vdash a, M, \Gamma \Rightarrow v, s \quad uop\ v = v'}{E \vdash uop\ a, M, \Gamma \Rightarrow v', s}$$

$$RV_{BOP}\ \frac{E \vdash a_1, M, \Gamma \Rightarrow v_1, s_1 \quad E \vdash a_2, M, \Gamma \Rightarrow v_2, s_2 \quad v_1\ bop\ v_2 = v \quad s_1 \sqcup s_2 = s}{E \vdash a_1\ bop\ a_2, M, \Gamma \Rightarrow v, s}$$

Figure 3: Sémantique étendue (cas des expressions).

des programmes vérifiés. Comme les locations des variables statiquement allouées sont à des décalages constants du pointeur de base, nous associons le label *public* aux valeurs gauches des variables (règle LV_{ID}). Le label de la valeur gauche de a définit le label associé à la valeur droite de $\&a$ (règle RV_{REF}). Le label de sécurité associé à la valeur droite de a est propagé à la valeur droite de $uop\ a$ (règle RV_{UOP}), où uop est un opérateur unaire. De manière similaire, l'union des labels associés aux valeurs droites de a_1 et a_2 est propagée à la valeur droite de l'expression $a_1\ bop\ a_2$ (règle RV_{bop}), où bop est un opérateur binaire.

La figure 4 illustre un exemple de l'évaluation en valeur droite de l'expression $*x$: si

$$RV\ \frac{LV_{MEM}\ \dfrac{RV\ \dfrac{LV_{ID}\ \dfrac{E(x) = l_x}{E \vdash x, M, \Gamma \Leftarrow l_x, public} \quad M(l_x) = ptr(l_a) \quad \Gamma(l_x) = s_x \quad s_x = public \sqcup s_x}{E \vdash x, M, \Gamma \Rightarrow ptr(l_a), s_x}}{E \vdash *x, M, \Gamma \Leftarrow l_a, s_x} \quad M(l_a) = v \quad \Gamma(l_a) = s_a \quad s = s_a \sqcup s_x}{E \vdash *x, M, \Gamma \Rightarrow v, s}$$

Figure 4: Exemple de l'évaluation d'une expression $*x$.

nous supposons que x est stockée à la location l_x et pointe vers une variable a stockée à la location l_a, l'évaluation droite de l'expression $*x$ prend alors en compte les flux d'information explicites et dûs au pointeurs puisque les deux labels s_x et s_a sont propagés au label de sécurité final $s = s_a \sqcup s_x$.

En conséquence des règles LV_{ID} et RV_{REF}, les adresses des variables sont considérées comme ayant le label *public*. Ces adresses peuvent donc être lues par les attaquants et utilisées pour compromettre des mesures de sécurité comme l'ASLR (randomisation de l'espace d'adressage). En fait, ce type de fuite d'information est hors de portée pour

l'analyse que nous mettons en œuvre puisque les adresses associées aux variables ne font pas partie des entrées secrètes du programme. Notons aussi que nous risquons un problème de *label creep* [15] si nous associons un label de sécurité autre que *public* aux valeurs gauches des variables. En effet, ce label de sécurité serait alors propagé à toutes les données accédées par le programme et interdirait donc toutes les sorties publiques.

$$(Assign)\ \frac{\begin{array}{c}E \vdash a_1, M, \Gamma \Leftarrow l_1, s_1 \quad E \vdash a_2, M, \Gamma \Rightarrow v_2, s_2 \\ s = s_1 \sqcup s_2 \sqcup \underline{pc} \quad s' = s_1 \sqcup \underline{pc} \quad M' = M[l_1 \mapsto v_2] \\ \Gamma'' = \Gamma[l_1 \mapsto s] \quad \Gamma' = update(a_1 = a_2, s', \Gamma'')\end{array}}{E \vdash a_1 = a_2, M, \Gamma, \underline{pc} \overset{\epsilon}{\Rightarrow} M', \Gamma'}$$

$$(Comp)\ \frac{\begin{array}{c}E \vdash c_1, M, \Gamma, \underline{pc} \overset{t_1}{\Rightarrow} M_1, \Gamma_1 \\ E \vdash c_2, M_1, \Gamma_1, \underline{pc} \overset{t_2}{\Rightarrow} M_2, \Gamma_2\end{array}}{E \vdash c_1; c_2, M, \Gamma, \underline{pc} \overset{t_1.t_2}{\Rightarrow} M_2, \Gamma_2}$$

$$(If_{tt})\ \frac{\begin{array}{c}E \vdash a, M, \Gamma \Rightarrow v, s \quad istrue(v) \\ \underline{pc}' = s \sqcup \underline{pc} \quad E \vdash c_1, M, \Gamma, \underline{pc}' \overset{t_1}{\Rightarrow} M_1, \Gamma_1 \\ \Gamma_1' = update(c_2, \underline{pc}', \Gamma_1)\end{array}}{E \vdash if\ (a)\ c_1\ else\ c_2, M, \Gamma, \underline{pc} \overset{t_1}{\Rightarrow} M_1, \Gamma_1'}$$

$$(W_{ff})\ \frac{\begin{array}{c}E \vdash a, M, \Gamma \Rightarrow v, s \quad isfalse(v) \\ \underline{pc}' = s \sqcup \underline{pc} \quad \Gamma' = update(c, \underline{pc}', \Gamma)\end{array}}{E \vdash while\ (a)\ c, M, \Gamma, \underline{pc} \overset{\epsilon}{\Rightarrow} M, \Gamma'}$$

$$(If_{ff})\ \frac{\begin{array}{c}E \vdash a, M, \Gamma \Rightarrow v, s \quad isfalse(v) \\ \underline{pc}' = s \sqcup \underline{pc} \quad E \vdash c_2, M, \Gamma, \underline{pc}' \overset{t_2}{\Rightarrow} M_2, \Gamma_2 \\ \Gamma_2' = update(c_1, \underline{pc}', \Gamma_2)\end{array}}{E \vdash if\ (a)\ c_1\ else\ c_2, M, \Gamma, \underline{pc} \overset{t_2}{\Rightarrow} M_2, \Gamma_2'}$$

$$(W_{tt})\ \frac{\begin{array}{c}E \vdash a, M, \Gamma \Rightarrow v, s \quad istrue(v) \\ \underline{pc}' = s \sqcup \underline{pc} \\ E \vdash c, M, \Gamma, \underline{pc}' \overset{t}{\Rightarrow} M', \Gamma' \\ E \vdash while\ (a)\ c, M', \Gamma', \underline{pc} \overset{t'}{\Rightarrow} M'', \Gamma''\end{array}}{E \vdash while\ (a)\ c, M, \Gamma, \underline{pc} \overset{t.t'}{\Rightarrow} M'', \Gamma''}$$

$$(Skip)\ E \vdash skip, M, \Gamma, \underline{pc} \Rightarrow M, \Gamma \qquad update(c, s, \Gamma) \triangleq \begin{cases}\Gamma(l) & \forall l \notin S_P(c) \\ \Gamma(l) \sqcup s & \forall l \in S_P(c)\end{cases}$$

$$(Out)\ \frac{E \vdash a, M, \Gamma \Rightarrow v, s' \quad s' \sqcup \underline{pc} \sqsubseteq s}{E \vdash \text{output}_s\ a, M, \Gamma, \underline{pc} \overset{o_s(v)}{\Rightarrow} M, \Gamma} \qquad (Supp)\ \frac{E \vdash a, M, \Gamma \Rightarrow v, s' \quad s' \sqcup \underline{pc} \not\sqsubseteq s}{E \vdash \text{output}_s\ a, M, \Gamma, \underline{pc} \overset{\epsilon}{\Rightarrow} M, \Gamma}$$

Figure 5: Sémantique étendue à grand pas (cas des instructions).

Instructions Nous présentons la sémantique des instructions dans la figure 5. Cette sémantique combine analyse dynamique et analyse statique grâce à l'ensemble $S_P(c)$ des locations qui pourraient être écrites par l'instruction c du programme P. Le moniteur fait appel à cet ensemble $S_P(c)$ à chaque fois que l'opérateur *update* est utilisé. Nous introduisons aussi une méta-variable $\underline{pc}$ permettant de capturer les flux implicites. $\underline{pc}$ peut être considéré comme un label de sécurité associé au compteur ordinal du programme vérifié. Dès que l'exécution d'un programme arrive sur une conditionnelle, $\underline{pc}$ est mis à jour grâce à la propagation du label de sécurité associé à la garde conditionnelle pour prendre en compte les flux implicites que cette garde génère. Ainsi, les instructions sont évaluées dans une mémoire Γ, un contexte d'exécution $\underline{pc}$ et une mémoire M pour produire de nouvelles mémoires Γ' et M' ainsi qu'une trace t. Les éléments de la trace t, notés sous la forme $o_s(v)$, correspondent à l'observation d'une valeur v sur un canal de niveau de sécurité $s \in \mathbb{S}$ suite à l'exécution d'une instruction $\text{output}_s\ a$.

La règle d'affectation $a_1 = a_2$ (règle *Assign*) associe l'union de trois labels de sécurité

$$(Assign)\ \frac{E \vdash *x, M, \Gamma \Leftarrow l_a, s_x \quad E \vdash 1, M, \Gamma \Rightarrow 1, public \quad s = s_x \sqcup public \sqcup \underline{pc} \quad s' = s_x \sqcup \underline{pc} \quad M' = M[l_a \mapsto 1] \quad \Gamma'' = \Gamma[l_a \mapsto s] \quad \Gamma' = update(*x = 1, s', \Gamma'')}{E \vdash *x = 1, M, \Gamma, \underline{pc} \overset{\epsilon}{\Rightarrow} M', \Gamma'}$$

Figure 6: Exemple d'évaluation d'une instruction $*x = 1$.

à la location l_1 de a_1. En effet, le premier label de sécurité s_1 prend en compte les flux d'information de a_1 dus aux pointeurs. Le second label de sécurité s_2 prend en compte les flux d'informations explicites et dus aux pointeurs provenant de a_2. Le troisième label de sécurité $\underline{pc}$ prend en compte les flux d'information implicites générés par les conditionnelles. Aussi, l'affectation $a_1 = a_2$ génère des flux d'information de la valeur droite de a_1 vers l'ensemble des locations qui auraient pu être écrites par cette affectation : l'opérateur *update* propage donc l'union des labels pc et s_1 vers l'ensemble $S_P(a_1 = a_2)$. En supposant que x pointe vers une variable a dont la location est notée l_a, la figure 6 illustre un exemple d'évaluation de l'instruction $*x = 1$. La label de sécurité s_x (resp. $\underline{pc}$) est propagé au label de sécurité associé à la location l_a afin de prendre en compte les flux d'information dus aux pointeurs (resp. les flux implicites). Aussi, l'opérateur *update* propage le label de sécurité s' vers l'ensemble $S_P(*x = 1)$ pour prendre en compte les flux d'information dus au pointeurs qui sont générés par l'affectation $*x = 1$.

Pour les conditionnelles (règle If_{tt} et If_{ff}), un nouveau contexte d'exécution $\underline{pc'}$ prend en compte les flux implicites générés par la garde conditionnelle. Si a est évaluée à vrai (la règle If_{ff} est symétrique), la mémoire de sécurité qui résulte de l'exécution de la conditionnelle prend en compte les flux d'information implicites générés par la branche exécutée c_1 ainsi que la branche non-exécutée c_2. Les flux implicites générés par c_1 sont pris en compte grâce à l'évaluation de c_1 dans le nouveau contexte d'exécution $\underline{pc'}$, alors que les flux implicites générés par c_2 sont pris en compte par l'opérateur *update* qui propage pc' à l'ensemble $S_P(c_2)$. Les règles W_{tt} et W_{ff} sont similaires aux conditionnelles. L'instruction $\text{output}_s\ a$ d'affichage/sortie (règles *Out* et *Supp*) génère une trace observable $o_s(v)$ si et seulement si l'observation de la valeur v sur le canal s ne viole pas la confidentialité ($s' \sqsubseteq s$) et si l'observation d'une quelconque valeur ne délivre aucune information sensible concernant les gardes conditionnelles ($\underline{pc} \sqsubseteq s$). Enfin, une séquence d'instruction $c_1; c_2$ est simplement exécutée dans le même contexte d'exécution (règle *Comp*).

Non-interférence Afin de formaliser la propriété de non-interférence insensible à la terminaison, nous introduisons en définition 1 une relation d'équivalence sur les mémoires M : deux mémoires M_1 et M_2 sont s-équivalentes si et seulement si les valeurs $M_1(l)$ et $M_2(l)$ sont égales pour toutes les zones mémoires l dont le label de sécurité est au plus égale à s ($\Gamma(l) \sqsubseteq s$).

Définition 1 (Relation d'équivalence $\sim^s_\Gamma$) *Pour tout Γ, $s \in \mathbb{S}$, M_1, M_2, M_1 et M_2 sont s-équivalentes ($M_1 \sim^s_\Gamma M_2$) si et seulement si*

$$\forall l \in Loc, \Gamma(l) \sqsubseteq s \implies M_1(l) = M_2(l).$$

Notons qu'un attaquant ayant une habilitation $s \in \mathbb{S}$ n'observe que les valeurs affichées sur des canaux s' tels que $s' \sqsubseteq s$. Nous introduisons donc en définition 2 un opérateur

$\Pi_s(t)$ permettant de projeter une trace t sur l'ensemble des valeurs observables par un attaquant ayant les droits $s \in \mathbb{S}$.

Définition 2 (Trace observable $\Pi_s(t)$ par un attaquant s)
Pour toute trace t constituée d'une séquence d'observations sous la forme $o_{s'}(v)$, la trace observable $\Pi_s(t)$ par un attaquant s est définie comme la sous-séquence contenant tous les éléments et seulement les éléments de t tels que $s' \sqsubseteq s$.

La non-interférence, selon la définition 3, assure qu'un attaquant connaissant seulement les entrées et sorties au plus à un niveau de sécurité s n'acquiert aucune connaissance concernant les entrées du programme dont le label de sécurité est strictement plus grand que s. En effet, deux exécutions à partir de deux mémoires s-équivalentes fournissent exactement la même trace observable $\Pi_s(t)$ pourvu que ces deux exécutions terminent.

Définition 3 (Non-interférence insensible à la terminaison)
Pour tout $c, E, \Gamma, M_1, M_1', M_2, M_2'$, pour tout $s, \underline{pc} \in \mathbb{S}$, tels que
$E \vdash c, M_1, \Gamma, \underline{pc} \overset{t_1}{\Rightarrow} M_1', \Gamma_1'$ *et* $E \vdash c, M_2, \Gamma, \underline{pc} \overset{t_2}{\Rightarrow} M_2', \Gamma_2'$, *alors*

$$M_1 \sim_\Gamma^s M_2 \implies \Pi_s(t_1) = \Pi_s(t_2)$$

Cette définition de la non-interférence est insensible à la terminaison puisqu'elle ignore complètement la divergence des programmes. Ainsi, les possibles fuites d'information dues à l'observation de la divergence par l'attaquant sont ignorées. Cette définition de TINI est équivalente aux définitions présentes dans la littérature [5, 6, 7, 8, 9, 10].

Théorème 1 (Correction) *La sémantique du moniteur de flux d'information est correcte vis-à-vis de la propriété de non-interférence insensible à la terminaison définie en 3.*

Le théorème 1 prouve que la sémantique du moniteur de flux d'information est correcte vis-à-vis de la propriété de non-interférence telle que définie en 3. La preuve, par induction sur l'évaluation des instructions $\Rightarrow$, s'appuie principalement sur le fait que l'évaluation gauche (resp. droite) des expressions dans des mémoires s-équivalentes fournit la même location (resp. valeur) pour les expressions dont le label de sécurité est au plus s. Elle s'articule en deux phases :

- Une preuve par induction d'une propriété de *batch-job* TINI [6] : les mémoires de sécurité finales sont égales pour les deux exécutions ($\Gamma = \Gamma_1' = \Gamma_2'$) et les mémoires de valeurs finales sont s-équivalentes ($M_1' \sim_\Gamma^s M_2'$).
- Une preuve par induction qu'un attaquant s observe la même trace pour les deux exécutions ($\Pi_s(t_1) = \Pi_s(t_2)$).

4 Travaux connexes

Le Guernic et al. [5] formalisent un moniteur de flux d'information, pour un langage impératif simple incluant des instructions d'affichage/sortie mais pas de pointeurs. Ce moniteur sensible aux flux de données combine analyse statique et dynamique pour mettre en œuvre la non-interférence de manière correcte. Il est formalisé grâce à un automate qui supprime les sorties secrètes du programme sur des canaux publiques ou les remplace

par des valeurs par défaut pour que l'exécution soit conforme à la non-interférence. Nous adoptons une approche différente qui consiste à toujours supprimer les sorties secrètes sur des canaux publiques afin d'éviter que le comportement de notre moniteur ne cause des fuites d'information. Le Guernic [16] étend aussi ce moniteur à un langage concurrent prenant en compte la synchronisation.

Russo et Sabelfeld [8] formalisent un moniteur hybride de flux d'information pour un langage impératif simple ne gérant pas les pointeurs. Ce moniteur est paramétré par le type d'action à entreprendre lorsqu'une exécution est interférente : remplacer la sortie par une valeur par défaut, arrêter le programme, ou supprimer l'affichage/sortie. Russo et Sabelfeld prouvent aussi la nécessité de reposer sur une analyse statique afin de correctement mettre en œuvre la non-interférence pour un moniteur sensible aux flux. Contrairement à Russo et Sabelfeld, nous formalisons notre moniteur en utilisant une sémantique à grands pas. Cette approche a l'avantage de simplifier considérablement la sémantique du moniteur : il n'y a nul besoin de maintenir une pile de labels de sécurité pour le contexte d'exécution $\underline{pc}$, ni de notifier le moniteur à chaque fois que l'exécution arrive sur le post-dominateur immédiat d'une instruction conditionnelle.

Moore et Chong [10] étendent le moniteur de Russo et Sabelfeld avec des références allouées dynamiquement, tout en permettant différentes abstractions de la mémoire. Notre approche peut être vue comme l'instantiation la plus précise de leur abstraction de la mémoire, où chaque location concrète est représentée par une seule location abstraite. Dans le cas le plus général, il n'est pas décidable de déterminer l'ensemble des locations qui pourraient être écrites par une instruction. Cependant, nous soutenons que pour des raisons de permissivité, il est nécessaire d'être le plus précis possible, au moins pour l'ensemble fini des locations statiquement allouées. Moore et Chong améliorent aussi les performances de leur moniteur grâce à une analyse statique permettant de limiter le nombre de variables à surveiller afin de réduire le surcoût à l'exécution.

Austin et Flanagan [7] proposent un moniteur purement dynamique pour un λ-calcul comprenant des références. Leur moniteur utilise une forme limitée de sensibilité aux flux de données puisqu'il implémente une politique conservative interdisant au programme d'écrire dans une variable publique lorsque le contexte d'exécution est secret (*no-sensitive upgrade policy*). Grâce à cette politique, ce moniteur est prouvé correct vis-à-vis de la non-interférence sans avoir à utiliser d'analyse statique. Austin et Flanagan [9] améliorent aussi leur moniteur grâce à une politique un peu plus permissive : lorsqu'une variable publique est écrite dans un contexte d'exécution secret, le moniteur la marque comme ayant partiellement fuité et donc interdit tout branchement sur cette variable. Le moniteur que nous proposons est complètement sensible aux flux de données. Il assure donc plus de permissivité tout en étant correct vis-à-vis de la non-interférence.

5 Conclusion

Nous avons formalisé un moniteur hybride de flux d'information combinant analyse statique et analyse dynamique. Ce moniteur est permissif puisqu'il est sensible aux flux de données. Il est aussi prouvé correct vis-à-vis de TINI. La sémantique de notre moniteur ignore la non-terminaison des programmes puisqu'elle est basée sur une version simplifiée de la sémantique à grands pas du langage Clight, ne prenant pas en compte la coinduction [11]. Ceci n'est toutefois pas problématique comme le font remarquer Le Guer-

nic et al. [5], puisque les fuites d'informations dues à la non-terminaison sont complètement ignorées lorsque nous nous intéressons à TINI.

Nous comptons étendre la sémantique de notre moniteur pour supporter un sous-ensemble plus large du langage C. Nous comptons aussi utiliser l'analyse de valeur de la plateforme Frama-C [17] afin de calculer une approximation correcte de l'ensemble $S_P(c)$ des locations écrites par une instruction c. Un prototype d'un moniteur hybride de flux d'information pour le langage C est en cours d'implémentation.

Références

[1] Denning, D., Denning, P. : Certification of Programs for Secure Information Flow. Communications of the ACM **20**(7) (1977) 504–513

[2] Goguen, J., Meseguer, J. : Security Policies and Security Models. In : IEEE Symposium on Research in Security and Privacy. (1982)

[3] Volpano, D., Irvine, C., Smith, G. : A Sound Type System for Secure Flow Analysis. Journal in Computer Security **4**(2-3) (1996) 167–187

[4] Hunt, S., Sands, D. : On Flow-Sensitive Security Types. In : Conference record of the 33rd ACM SIGPLAN-SIGACT symposium on Principles of programming languages. Volume 41., ACM (2006) 79–90

[5] Le Guernic, G., Banerjee, A., Jensen, T., Schmidt, D. : Automata-Based Confidentiality Monitoring. Advances in Computer Science-ASIAN 2006. Secure Software and Related Issues (2007) 75–89

[6] Askarov, A., Hunt, S., Sabelfeld, A., Sands, D. : Termination-Insensitive Noninterference Leaks More Than Just a Bit. In : Computer Security - ESORICS 2008. Volume 5283 of Lecture Notes in Computer Science. (2008)

[7] Austin, T., Flanagan, C. : Efficient Purely-Dynamic Information Flow Analysis. ACM Sigplan Notices **44**(8) (2009) 20–31

[8] Russo, A., Sabelfeld, A. : Dynamic vs. Static Flow-Sensitive Security Analysis. Computer Security Foundations Symposium, IEEE **0** (2010) 186–199

[9] Austin, T.H., Flanagan, C. : Permissive Dynamic Information Flow Analysis. In : PLAS '10 : Proceedings of the 5th ACM SIGPLAN Workshop on Programming Languages and Analysis for Security, ACM (2010) 1–12

[10] Moore, S., Chong, S. : Static Analysis for Efficient Hybrid Information-Flow Control. In : Computer Security Foundations Symposium (CSF), 2011 IEEE 24th, IEEE (2011) 146–160

[11] Blazy, S., Leroy, X. : Mechanized Semantics for the Clight Subset of the C Language. Journal of Automated Reasoning **43**(3) (2009) 263–288

[12] Leroy, X. : Formal Verification of a Realistic Compiler. Communications of the ACM **52**(7) (2009) 107–115

[13] Assaf, M., Signoles, J., Tronel, F., Totel, E. : Program Transformation for Noninterference Verification on Programs with Pointers. Research report RR-8284, INRIA (April 2013) URL : http ://hal.inria.fr/hal-00814671, en cours de publication.

[14] Kahn, G. : Natural semantics. In : 4th Annual Symposium on Theoretical Aspects of Computer Sciences on STACS 87, London, UK, UK, Springer-Verlag (1987) 22–39

[15] Sabelfeld, A., Myers, A. : Language-Based Information-Flow Security. Selected Areas in Communications, IEEE Journal on **21**(1) (2003) 5–19

[16] Le Guernic, G. : Automaton-based confidentiality monitoring of concurrent programs. In : Computer Security Foundations Symposium, 2007. CSF'07. 20th IEEE, IEEE (2007) 218–232

[17] Cuoq, P., Kirchner, F., Kosmatov, N., Prevosto, V., Signoles, J., Yakobowski, B. : Frama-C : A Program Analysis Perspective. Software Engineering and Formal Methods (2012) 233–247

We prove that our monitor is sound with respect to TINI. We take advantage of the symmetry of both runs by introducing a partial order relation 5 on security memories Γ. Definition 6 is equivalent to Definition 4 when both input security memories and both execution contexts are equal in both runs.

Définition 4 (Termination-insensitive non-interference)
For all $c, E, \Gamma, M_1, M_1', M_2, M_2', s, \underline{pc} \in \mathbb{S}$, *such that*
$E \vdash c, M_1, \Gamma, \underline{pc} \Rightarrow M_1', \Gamma_1'$ *and* $E \vdash c, M_2, \Gamma, \underline{pc} \Rightarrow M_2', \Gamma_2'$,

$$M_1 \sim_\Gamma^s M_2 \implies \Gamma_2' = \Gamma_1' = \Gamma' \text{ and } M_1' \sim_{\Gamma'}^s M_2'.$$

Définition 5 (Less restrictive up to label s ($\sqsubseteq_s$))
For all $s \in \mathbb{S}$, *all* Γ_1, Γ_2, Γ_2 *is less restrictive than* Γ_1 *up to security label* s *(*$\Gamma_2 \sqsubseteq_s \Gamma_1$*) iff for all* $l \in Loc$,

$$\Gamma_1(l) \sqsubseteq s \text{ implies } \Gamma_2(l) \sqsubseteq \Gamma_1(l)$$

Définition 6 (Batch-job termination-insensitive non-interference)
For all $c, E, \Gamma_1, \Gamma_2, M_1, M_1', M_2, M_2'$, *for all* $s, \underline{pc}_1, \underline{pc}_2 \in \mathbb{S}$, *such that*
$E \vdash c, M_1, \Gamma_1, \underline{pc}_1 \Rightarrow M_1', \Gamma_1'$ *and* $E \vdash c, M_2, \Gamma_2, \underline{pc}_2 \Rightarrow M_2', \Gamma_2'$, *then*

$$\underline{pc}_2 \sqsubseteq \underline{pc}_1 \text{ and } \Gamma_2 \sqsubseteq_s \Gamma_1 \text{ and } M_1 \sim_{\Gamma_1}^s M_2 \text{ implies } \Gamma_2' \sqsubseteq_s \Gamma_1' \text{ and } M_1' \sim_{\Gamma_1'}^s M_2'$$

We introduce Lemma 1, which proves that for two s-equivalent memories, if the l-value evaluation of an expression yields a label below s, then it is evaluated to the same location in the second run, and to a label that is less restrictive than the label of the first run.

Lemme 1 (L-value evaluation in s-equivalent memories)
For all $E, M_1, M_2, \Gamma_1, \Gamma_2, s \in \mathbb{S}$, *such that* $\Gamma_2 \sqsubseteq_s \Gamma_1$ *and* $M_1 \sim_{\Gamma_1}^s M_2$,
for all $a \in Exp$ *such that* $E \vdash a, M_1, \Gamma_1 \Leftarrow l_1, s_1$ *and* $E \vdash a, M_2, \Gamma_2 \Leftarrow l_2, s_2$
then
$s_1 \sqsubseteq s$ *implies that* $l_1 = l_2$ *and* $s_2 \sqsubseteq s_1$

Proof:
Let: $E, M_1, M_2, \Gamma_1, \Gamma_2$ and $s \in \mathbb{S}$.
Let: $a \in Exp$ such that $E \vdash a, M_1, \Gamma_1 \Leftarrow l_1, s_1$ and $E \vdash a, M_2, \Gamma_2 \Leftarrow l_2, s2$.
Assume: 1. $\Gamma_2 \sqsubseteq_s \Gamma_1$
2. $M_1 \sim_{\Gamma_1}^s M_2$
3. $s_1 \sqsubseteq s$
Prove: 1. $l_1 = l_2$
2. $s_2 \sqsubseteq s_1$
Proof sketch: By induction on l-value evaluations of expressions.
⟨1⟩1. Case: LV_{id}
⟨2⟩1. $s_2 = s_1 = public$
⟨3⟩1. Q.E.D.
Proof: By rule LV_{id}, labels associated to l-values of variables are defined as the bottom of $\mathbb{S}$

$\langle 2\rangle 2.\ E(a) = l_1 = l_2$
$\langle 3\rangle 1.$ Q.E.D.
PROOF: By rule LV_{id}. Environment E is the same for both runs.
$\langle 2\rangle 3.$ Q.E.D.
PROOF: By $\langle 2\rangle 1$ and $\langle 2\rangle 2$.
$\langle 1\rangle 2.$ CASE: LV_{MEM}
LET: l_a, s_l, l, s_r and l'_a, s'_l, l', s'_r such that

$$LV_{MEM}\ \dfrac{RV\ \dfrac{(\mathbf{1})E \vdash a, M_1, \Gamma_1 \Leftarrow l_a, s_l \qquad M_1(l_a) = ptr(l_1) \qquad \Gamma_1(l_a) = s_r \qquad s_1 = s_l \sqcup s_r}{E \vdash a, M_1, \Gamma_1 \Rightarrow ptr(l_1), s_1}}{E \vdash *a, M_1, \Gamma_1 \Leftarrow l_1, s_1}$$

$$LV_{MEM}\ \dfrac{RV\ \dfrac{E \vdash a, M_2, \Gamma_2 \Leftarrow l'_a, s'_l \qquad M_2(l'_a) = ptr(l_2) \qquad \Gamma_2(l'_a) = s'_r \qquad s_2 = s'_l \sqcup s'_r}{E \vdash a, M_2, \Gamma_2 \Rightarrow ptr(l_2), s_2}}{E \vdash *a, M_2, \Gamma_2 \Leftarrow l_2, s_2}$$

SUFFICES ASSUME: $s_1 \sqsubseteq s$
PROVE: 1. $s_2 \sqsubseteq s_1$
2. $l_2 = l_1$
$\langle 2\rangle 1.$ $s_l \sqsubseteq s$ and $s_r \sqsubseteq s$
$\langle 3\rangle 1.$ Q.E.D.
PROOF: By assumption of $\langle 1\rangle 2$ $s_1 = s_l \sqcup s_r \sqsubseteq s$
$\langle 2\rangle 2.$ $l'_a = l_a$ and $s'_l \sqsubseteq s_l$
$\langle 3\rangle 1.$ Q.E.D.
PROOF: By induction on derivation depth of $*a$, using $\langle 2\rangle 1$ and assumptions 1 and 2
$\langle 2\rangle 3.$ $s'_r \sqsubseteq s_r$
$\langle 3\rangle 1.$ Q.E.D.
PROOF: By $\langle 2\rangle 1$ and assumption 1 : $\Gamma_2(l_a) \sqsubseteq \Gamma_1(l_a) \sqsubseteq s$
$\langle 2\rangle 4.$ Q.E.D.
PROOF: By $\langle 2\rangle 2$ and $\langle 2\rangle 3$, $s_2 = s'_l \sqcup s'_r \sqsubseteq s_l \sqcup s_r = s_1$, and $l_1 = l_2$ by assumption 2 and $\langle 2\rangle 1$.
$\langle 1\rangle 3.$ Q.E.D.
PROOF: By induction on the evaluation of l-values and $\langle 1\rangle 1$ and $\langle 1\rangle 2$.

Corollary 1 generalizes the result of Lemma 1 to include all expressions.

Corollaire 1 (R-value evaluation in s-equivalent memories)
For all $E, M_1, M_2, \Gamma_1, \Gamma_2, s \in \mathbb{S}$, such that $\Gamma_2 \sqsubseteq_s \Gamma_1$ and $M_1 \sim^s_{\Gamma_1} M_2$, for all $a \in Exp$ such that $E \vdash a, M_1, \Gamma_1 \Rightarrow v_1, s_1$ and $E \vdash a, M_2, \Gamma_2 \Rightarrow v_2, s_2$ then
$s_1 \sqsubseteq s$ implies that $v_1 = v_2$ and $s_2 \sqsubseteq s_1$

$\langle 1\rangle 1.$ Q.E.D.
PROOF: By induction on r-value evaluations, using lemma 1.

Theorem 1 proves the soundness of our semantics with respect to TINI.

Théorème 1 (Correction) *La sémantique du moniteur de flux d'information est correcte vis-à-vis de la propriété de non-interférence insensible à la terminaison définie en 3.*

PROOF:
By induction on the evaluation of instructions.
LET: $E, \Gamma_1, \Gamma_2, M_1, M_1', M_2, M_2'$ and $\underline{pc}_1, \underline{pc}_2 \in \mathbb{S}$ such that :
$E \vdash c, M_1, \Gamma_1, \underline{pc}_1 \Rightarrow M_1', \Gamma_1'$ and $E \vdash c, M_2, \Gamma_2, \underline{pc}_2 \Rightarrow M_2', \Gamma_2'$.
LET: Let $s \in \mathbb{S}$.

ASSUME: 1. $\underline{pc}_2 \sqsubseteq \underline{pc}_1$
2. $\Gamma_2 \sqsubseteq_s \Gamma_1$
3. $M_1 \sim^s_{\Gamma_1} M_2$
PROVE: 1. $\Gamma_2' \sqsubseteq_s \Gamma_1'$
2. $M_1' \sim^s_{\Gamma_1'} M_2'$

⟨1⟩1. CASE: Skip
PROOF: By assumptions 2 and 3.
⟨1⟩2. CASE: Assign
LET: $l_1, s_1, v_2, s_2, l_1', s_1', v_2', s_2'$ such that the evaluation of instruction $a_1 = a_2$ in both M_1, Γ_1 and M_2, Γ_2 yield :

$$(Assign)\ \frac{\begin{array}{c}E \vdash a_1, M_1, \Gamma_1 \Leftarrow l_1, s_1 \quad E \vdash a_2, M_1, \Gamma_1 \Rightarrow v_2, s_2 \quad s_{l_1} = s_1 \sqcup s_2 \sqcup \underline{pc}_1 \\ M_1' = M_1[l_1 \hookleftarrow v_2] \quad \Gamma_1'' = \Gamma_1[l_1 \hookleftarrow s_{l_1}] \quad \Gamma_1' = update(a_1 = a_2, s_1, \Gamma_1'')\end{array}}{E \vdash a_1 = a_2, M_1, \Gamma_1, \underline{pc}_1 \Rightarrow M_1', \Gamma_1'}$$

$$(Assign)\ \frac{\begin{array}{c}E \vdash a_1, M_2, \Gamma_2 \Leftarrow l_1', s_1' \quad E \vdash a_2, M_2, \Gamma_2 \Rightarrow v_2', s_2' \quad s_{l_1}' = s_1' \sqcup s_2' \sqcup \underline{pc}_2 \\ M_2' = M_2[l_1' \hookleftarrow v_2'] \quad \Gamma_2'' = \Gamma_2[l_1' \hookleftarrow s_{l_1}'] \quad \Gamma_2' = update(a_1 = a_2, s_1', \Gamma_2'')\end{array}}{E \vdash a_1 = a_2, M_2, \Gamma_2, \underline{pc}_2 \Rightarrow M_2', \Gamma_2'}$$

SUFFICES ASSUME: $l \in \Gamma_1'^{-1}(s)$
PROVE: 1. $l \in \Gamma_2'^{-1}(s)$
2. $M_1'(l) = M_2'(l)$

⟨2⟩1. CASE: $l \in S_p(a_1 = a_2)$
PROOF SKETCH: In this case, the same location $l_1 = l_1'$ is modified by the assignment $a_1 = a_2$ for both runs. It suffices to consider two cases :
— $l = l_1$: the initial values mapped to l are modified in both runs. Yet, the same values are mapped to l after the execution of the assignment since its output security label is below s
— $l \neq l_1$: only the security label of l is modified by the update operator. Yet, it is still below s for both runs.
⟨3⟩1. $l_1 = l_1'$ and $s_1' \sqsubseteq s$
⟨4⟩1. $s_1 \sqsubseteq s$
⟨5⟩1. $\Gamma_1'(l) = \Gamma_1''(l) \sqcup s_1$
⟨5⟩2. Q.E.D.
PROOF: By assumption of ⟨1⟩2 ($\Gamma_1'(l) \sqsubseteq s$)
⟨4⟩2. Q.E.D.
PROOF: by ⟨4⟩1, and Lemma 1 on the l-value evaluation of a_1 in both M_1, Γ_1 and M_2, Γ_2, and assumptions 2 and 3.

⟨3⟩2. CASE: $l = l_1$
⟨4⟩1. $\Gamma'_1(l_1) = s_1 \sqcup s_2 \sqcup \underline{pc}_1 \sqsubseteq s$
⟨4⟩2. $v_2 = v'_2$ and $s'_2 \sqsubseteq s$
⟨5⟩1. $s_2 \sqsubseteq s$
⟨6⟩1. Q.E.D.
PROOF: By ⟨4⟩1
⟨5⟩2. Q.E.D.
PROOF: By ⟨5⟩1, and Corollary 1 on the r-value evaluation of a_2 in both M_1, Γ_1 and M_2, Γ_2, and assumptions 2 and 3.
⟨4⟩3. $\Gamma'_2(l_1) = s'_1 \sqcup s'_2 \sqcup \underline{pc}_2 \sqsubseteq s$
⟨5⟩1. $\underline{pc}_2 \sqsubseteq s$
⟨6⟩1. Q.E.D.
PROOF: By ⟨4⟩1 and assumption 1
⟨5⟩2. Q.E.D.
PROOF: By ⟨3⟩1 and ⟨4⟩2 and ⟨4⟩3
⟨4⟩4. Q.E.D.
PROOF: By ⟨4⟩3 and ⟨4⟩2
⟨3⟩3. CASE: $l \neq l_1$
⟨4⟩1. $\Gamma_1(l) \sqsubseteq s$
⟨5⟩1. $\Gamma'_1(l) = \Gamma''_1(l) \sqcup s_1$
⟨6⟩1. Q.E.D.
PROOF: By the semantics rule of the execution on M_1, Γ_1 and ⟨2⟩1
⟨5⟩2. $\Gamma''_1(l) = \Gamma_1(l)$
⟨6⟩1. Q.E.D.
PROOF: By ⟨3⟩3 and the semantic rule on M_1, Γ_1
⟨5⟩3. Q.E.D.
PROOF: By assumption of ⟨1⟩2 and ⟨5⟩1 and ⟨5⟩2
⟨4⟩2. $M'_1(l) = M'_2(l)$
⟨5⟩1. $M'_1(l) = M_1(l)$
⟨6⟩1. Q.E.D.
PROOF: By ⟨3⟩3
⟨5⟩2. $M'_2(l) = M_2(l)$
⟨6⟩1. Q.E.D.
PROOF: By ⟨3⟩3 and ⟨3⟩1
⟨5⟩3. $M_1(l) = M_2(l)$
⟨6⟩1. Q.E.D.
PROOF: By ⟨4⟩1 and assumptions 3 and 2
⟨5⟩4. Q.E.D.
PROOF: By ⟨5⟩1 and ⟨5⟩2 and ⟨5⟩3
⟨4⟩3. $\Gamma'_2(l) \sqsubseteq s$
⟨5⟩1. $\Gamma_2(l) \sqsubseteq s$
⟨6⟩1. Q.E.D.
PROOF: By ⟨4⟩1 and assumption 2
⟨5⟩2. $\Gamma_2(l) = \Gamma''_2(l)$
⟨6⟩1. Q.E.D.
PROOF: By ⟨3⟩3 and ⟨3⟩1

⟨5⟩3. $\Gamma'_2(l) = \Gamma''_2(l) \sqcup s'_1$
⟨6⟩1. Q.E.D.
PROOF: By the update operator of the semantic rule on M_2, Γ_2 and ⟨2⟩1
⟨5⟩4. Q.E.D.
PROOF: By ⟨5⟩1, ⟨5⟩2, ⟨5⟩3 and ⟨3⟩1
⟨4⟩4. Q.E.D.
PROOF: By ⟨4⟩3 and ⟨4⟩2
⟨2⟩2. CASE: $l \notin S_p(a_1 = a_2)$
PROOF SKETCH: In this case, neither the value nor the security label associated to the location l changes.
⟨3⟩1. ok
⟨4⟩1. Q.E.D.
PROOF: By ⟨2⟩2 and assumption of ⟨1⟩2
⟨3⟩2. $M'_1(l) = M'_2(l)$
⟨4⟩1. $M'_1(l) = M_1(l)$
⟨5⟩1. Q.E.D.
PROOF: By ⟨2⟩2 since $l_1 \neq l$
⟨4⟩2. $M'_2(l) = M_2(l)$
⟨5⟩1. Q.E.D.
PROOF: By ⟨2⟩2 since $l'_1 \neq l$
⟨4⟩3. $M_1(l) = M_2(l)$
⟨5⟩1. Q.E.D.
PROOF: By ⟨3⟩1 and assumptions 2 and 3
⟨4⟩4. Q.E.D.
PROOF: By ⟨4⟩3, ⟨4⟩2 and ⟨4⟩1
⟨3⟩3. $\Gamma'_2(l) \sqsubseteq s$
⟨4⟩1. $\Gamma'_2(l) = \Gamma_2(l)$
⟨5⟩1. Q.E.D.
PROOF: By ⟨2⟩2
⟨4⟩2. $\Gamma_2(l) \sqsubseteq s$
⟨5⟩1. Q.E.D.
PROOF: By ⟨3⟩1 and assumption 2
⟨4⟩3. Q.E.D.
PROOF: By ⟨4⟩2 and ⟨4⟩1
⟨3⟩4. Q.E.D.
PROOF: By ⟨3⟩3 and ⟨3⟩2
⟨2⟩3. Q.E.D.
PROOF:By ⟨2⟩1 and ⟨2⟩2
⟨1⟩3. CASE: If_{tt}
LET: v_1, s_1, v_2, s_2 such that the evaluation of instruction $if\ (a)\ c_1\ else\ c_2$ in M_1, Γ_1 yield :

$$(If_{tt})\ \frac{\begin{array}{c} E \vdash a, M_1, \Gamma_1 \Rightarrow v_1, s_1 \quad istrue(v_1) \\ \underline{pc}'_1 = s_1 \sqcup \underline{pc}_1 \quad E \vdash c_1, M_1, \Gamma_1, \underline{pc}'_1 \Rightarrow M'_1, \Gamma''_1 \\ \Gamma'_1 = update(c_2, \underline{pc}'_1, \Gamma''_1) \end{array}}{E \vdash if\ (a)\ c_1\ else\ c_2, M_1, \Gamma_1, \underline{pc}_1 \Rightarrow M'_1, \Gamma'_1} \qquad RV\ E \vdash a, M_2, \Gamma_2 \Rightarrow v_2, s_2$$

LET: $l \in Loc$
SUFFICES ASSUME: $\Gamma'_1(l) \sqsubseteq s$
PROVE: 1. $\Gamma'_2(l) \sqsubseteq \Gamma'_1(l)$
2. $M'_1(l) = M'_2(l)$

$\langle 2 \rangle 1$. CASE: $s_1 \sqsubseteq s$

PROOF SKETCH: In this case, both executions in M_1, Γ_1 and M_2, Γ_2 execute instruction c_1. Therefore, an induction on c_1 is sufficient.

$\langle 3 \rangle 1$. $v_2 = v_1$ and $s_2 \sqsubseteq s_1$

$\langle 3 \rangle 2$. Q.E.D.

PROOF: By assumption of $\langle 2 \rangle 1$ and Corollary 1.

Execution is M_2, Γ_2 yield :

$$If_{tt}\ \frac{\begin{array}{c} E \vdash a, M_2, \Gamma_2 \Rightarrow v_2, s_2 \quad istrue(v_2) \\ pc'_2 = s_2 \sqcup \underline{pc}_2 \quad E \vdash c_1, M_2, \Gamma_2 \underline{pc'_2} \Rightarrow M'_2, \Gamma''_2 \\ \Gamma'_1 = update(c_2, \underline{pc'_2}, \Gamma''_2) \end{array}}{E \vdash if\ (a)\ c_1\ else\ c_2, M_2, \Gamma_2, \underline{pc}_2 \Rightarrow M'_2, \Gamma'_2}$$

$\langle 3 \rangle 3$. $pc'_2 \sqsubseteq \underline{pc'_1}$

$\langle 4 \rangle 1$. Q.E.D.

PROOF: By $\langle 3 \rangle 1$ and assumption 1 and rules If_{tt} of both executions.

$\langle 3 \rangle 4$. $M'_1 \sim^s_{\Gamma_1} M'_2$ and $\Gamma''_2 \sqsubseteq_s \Gamma''_1$

$\langle 4 \rangle 1$. Q.E.D.

PROOF: By $\langle 3 \rangle 3$, assumptions 3 and 2 and induction hypothesis on the evaluation of instruction c_1.

$\langle 3 \rangle 5$. $\Gamma'_2 \sqsubseteq_s \Gamma'_1$

LET: $l \in loc$
SUFFICES ASSUME: $\Gamma'_1(l) \sqsubseteq s$
PROVE: $\Gamma'_2(l) \sqsubseteq \Gamma'_1(l)$

$\langle 4 \rangle 1$. CASE: $l \notin S_p(c_2)$

$\langle 5 \rangle 1$. Q.E.D.

PROOF: By $\langle 3 \rangle 4$, assumption of $\langle 4 \rangle 1$ and definition of operator update, $\Gamma'_1(l) = \Gamma''_2(l) \sqsubseteq \Gamma''_1(l) = \Gamma'_1(l)$

$\langle 4 \rangle 2$. CASE: $l \in S_p(c_2)$

$\langle 5 \rangle 1$. Q.E.D.

PROOF: By $\Gamma'_2(l) = \underline{pc'_2} \sqsubseteq \Gamma''_2(l)$ and $\Gamma'_1(l) = \underline{pc'_1} \sqsubseteq \Gamma''_1(l)$ and $\langle 3 \rangle 4$ and $\langle 3 \rangle 3$.

$\langle 4 \rangle 3$. Q.E.D.

PROOF: By cases $\langle 4 \rangle 1$ and $\langle 4 \rangle 2$

$\langle 3 \rangle 6$. Q.E.D.

PROOF: By $\langle 3 \rangle 5$ and $\langle 3 \rangle 4$.

$\langle 2 \rangle 2$. CASE: $s_1 \not\sqsubseteq s$

PROOF SKETCH: In this case, if the label of location l is below s, that means that location l could not have been assigned neither by instructions c_1, nor by instruction c_2. Otherwise, $\underline{pc}$ would have been propagated to the label of l which would be greater than s_1.

$\langle 3 \rangle 1$. $l \notin S_p(c_2)$

$\langle 4 \rangle 1$. Q.E.D.

PROOF: By assumption of $\langle 1 \rangle 3$ ($\Gamma'_1(l) \sqsubseteq s$). In fact, $l \in S_p(c_2)$ implies that $s_1 \sqsubseteq \Gamma'_1(l)$ which means $\Gamma'_1(l) \not\sqsubseteq s$.

⟨3⟩2. $l \notin S_P(c_1)$
⟨4⟩1. Q.E.D.
PROOF: If there exists an assignment in c_1 that may write to location l, then the update operator would propagate $\underline{pc}'_1 \not\sqsubseteq s$ to $\Gamma'_1(l)$.
⟨3⟩3. $M_1(l) = M'_1(l)$ and $\Gamma_1(l) = \Gamma'_1(l)$
⟨4⟩1. Q.E.D.
PROOF: By ⟨3⟩1 and ⟨3⟩2 since l is neither written by c_1 nor operator update.
⟨3⟩4. $M_2(l) = M'_2(l)$ and $\Gamma_2(l) = \Gamma'_2(l)$
⟨4⟩1. Q.E.D.
PROOF: By ⟨3⟩1 and ⟨3⟩2 since l is neither written by c_1, nor c_2.
⟨3⟩5. Q.E.D.
PROOF: By ⟨3⟩3 and ⟨3⟩4 and assumptions 2 and 3.
⟨2⟩3. Q.E.D.
PROOF: By ⟨2⟩1 and ⟨2⟩2.
⟨1⟩4. CASE: If_{ff}
PROOF: by symmetry of ⟨1⟩3.
⟨1⟩5. CASE: W_{tt} and W_{ff}
⟨2⟩1. Q.E.D.
PROOF: *While* rule is semantically equivalent to $if\ (a)\ c; while\ (a)\ c;\ else\ skip$
⟨1⟩6. CASE: Composition
⟨2⟩1. Q.E.D.
PROOF: By induction on c_1, then on c_2.
⟨1⟩7. Q.E.D.
PROOF: By ⟨1⟩1, ⟨1⟩2, ⟨1⟩3, ⟨1⟩4, ⟨1⟩5, ⟨1⟩6 and induction on instruction evaluation $\Rightarrow$.

Définition 3 (Non-interférence insensible à la terminaison)
Pour tout $c, E, \Gamma, M_1, M'_1, M_2, M'_2$, *pour tout* $s, \underline{pc} \in \mathbb{S}$, *tels que* $E \vdash c, M_1, \Gamma, \underline{pc} \overset{t_1}{\Rightarrow} M'_1, \Gamma'_1$ *et* $E \vdash c, M_2, \Gamma, \underline{pc} \overset{t_2}{\Rightarrow} M'_2, \Gamma'_2$, *alors*

$$M_1 \sim^s_\Gamma M_2 \implies \Pi_s(t_1) = \Pi_s(t_2)$$

PROOF: By induction on the evaluation of instructions.
LET: $E, \Gamma, M_1, M'_1, M_2, M'_2$ and $\underline{pc} \in \mathbb{S}$ such that :
$E \vdash c, M_1, \Gamma, \underline{pc} \overset{t_1}{\Rightarrow} M'_1, \Gamma'_1$ and $E \vdash c, M_2, \Gamma, \underline{pc} \overset{t_2}{\Rightarrow} M'_2, \Gamma'_2$.
LET: Let $s \in \mathbb{S}$
ASSUME: 1. $M_1 \sim^s_\Gamma M_2$
PROVE: 1. $|\Pi_s(t_1)| = |\Pi_s(t_2)|$
2. $\Pi_s(t_1) = \Pi_s(t_2)$
⟨1⟩1. $\Gamma'_1 = \Gamma'_2$ and $M'_1 \sim^s_{\Gamma'_1} M'_2$
⟨2⟩1. Q.E.D.
PROOF: By assumption 1 and Theorem 1.
⟨1⟩2. CASE: Skip, Assign
⟨2⟩1. Q.E.D.
PROOF: By $t_1 = t_2 = \epsilon$
⟨1⟩3. CASE: Composition

⟨2⟩1. Q.E.D.
PROOF: By induction on c_1, then by using ⟨1⟩1 for an induction on c_2.
⟨1⟩4. CASE: $\text{output}_l\ a$
⟨2⟩1. CASE: $l \not\sqsubseteq s$
PROOF SKETCH: In this case, anything that flows into channel l is not observable by attackers having a security level s.
⟨3⟩1. Q.E.D.
PROOF: Since in both runs, $E \vdash \text{output}_l\ a, M_i, \Gamma, \underline{pc} \overset{t_i}{\Rightarrow} M_i, \Gamma_i$ and $\Pi_s(t_i) = \epsilon$ by ⟨2⟩1 and definition of Π_s.
⟨2⟩2. CASE: $l \sqsubseteq s$
PROOF SKETCH: When attackers observe some values, they observe exactly the same values for both runs.
LET: v_i, s_i such that $E \vdash a, M_i, \Gamma \Rightarrow v_i, s_i$ for $i = 1, 2$.
⟨3⟩1. CASE: $(s_1 \sqcup \underline{pc} \not\sqsubseteq l)$
For both runs $i = 1, 2$, we have :

$$(NoOutput)\ \frac{E \vdash a, M_i, \Gamma \Rightarrow v_i, s_i \qquad s_i \sqcup \underline{pc} \not\sqsubseteq l}{E \vdash \text{output}_l\ a, M_i, \Gamma, \underline{pc} \overset{\epsilon}{\Rightarrow} M_i, \Gamma}$$

⟨4⟩1. Q.E.D.
PROOF: Either $\underline{pc} \not\sqsubseteq l$ or $s_1 \not\sqsubseteq l$. Both cases leads to the evaluation of rule *NoOutput* and hence $t_1 = t_2 = \epsilon$. Notice that when $s_1 \not\sqsubseteq l$, the contrapositive of corollary 1 (using $M_1 \sim^l_\Gamma M_2$) guarantees that $s_2 \not\sqsubseteq l$, and hence that the rule *NoOutput* is executed in both runs.
⟨3⟩2. CASE: $(s_1 \sqcup \underline{pc} \sqsubseteq l)$
For both runs $i = 1, 2$, we have :

$$(Output)\ \frac{E \vdash a, M_i, \Gamma \Rightarrow v_i, s \qquad s \sqcup \underline{pc} \sqsubseteq l}{E \vdash \text{output}_l\ a, M_i, \Gamma, \underline{pc} \overset{o_l(v_i)}{\Rightarrow} M_i, \Gamma}$$

⟨4⟩1. Q.E.D.
PROOF: By corollary 1, $s_1 \sqsubseteq l$ implies $s_2 \sqsubseteq l$ and $v_1 = v_2 = v$.
Hence, $\Pi_{t_1}(s) = \Pi_{t_2}(s) = v$.
⟨3⟩3. Q.E.D.
PROOF: By ⟨3⟩1 and ⟨3⟩2.
⟨2⟩3. Q.E.D.
PROOF: By ⟨2⟩1 and ⟨2⟩2.
⟨1⟩5. CASE: If_{tt}
LET: v_1, s_1 and v_1, s_2 such that : $E \vdash a, M_1, \Gamma \Rightarrow v_1, s_1$ and $E \vdash a, M_2, \Gamma \Rightarrow v_2, s_2$.
ASSUME: $istrue(v_1)$.
The evaluation in M_1, Γ yields :

$$If_{tt}\ \frac{\begin{array}{c} E \vdash a, M_1, \Gamma \Rightarrow v_1, s_1 \qquad istrue(v_1) \qquad \underline{pc}' = s_1 \sqcup \underline{pc} \\ E \vdash c_1, M_1, \Gamma, \underline{pc}' \overset{t_1}{\Rightarrow} M_1', \Gamma_1' \qquad \Gamma' = update(c_2, \underline{pc}', \Gamma_1') \end{array}}{E \vdash if\ (a)\ c_1\ else\ c_2, M_1, \Gamma, \underline{pc} \overset{t_1}{\Rightarrow} M_1', \Gamma'}$$

⟨2⟩1. CASE: $s_1 \sqsubseteq s$
⟨3⟩1. Q.E.D.
PROOF: By corollary 1, $s_2 = s_1$ and $v_2 = v_1$. That means that the rule If_{tt} also

applies for the execution in M_2, Γ, and we can finally conclude by induction on c_1.

$\langle 2 \rangle 2$. Case: $s_1 \not\sqsubseteq s$

$\langle 3 \rangle 1$. Q.E.D.

Proof: By the contrapositive of corollary 1, $s_2 \not\sqsubseteq s$. Hence $\underline{pc'_i} \not\sqsubseteq s$ for both runs $i = 1, 2$, and $\Pi_{t_1}(s) = \Pi_{t_2}(s) = \epsilon$.

$\langle 2 \rangle 3$. Q.E.D.

Proof: By $\langle 2 \rangle 1$ and $\langle 2 \rangle 2$.

$\langle 1 \rangle 6$. Case: If_{ff}, W_{tt} and W_{ff}

$\langle 2 \rangle 1$. Q.E.D.

Proof: By symmetry of the rule If_{tt}, the property holds for If_{ff}. Then it also holds for W_{ff} and W_{tt} since they are both semantically equivalent to a conditional statement.

$\langle 1 \rangle 7$. Q.E.D.

Proof: By $\langle 1 \rangle 2$, $\langle 1 \rangle 3$, $\langle 1 \rangle 4$, $\langle 1 \rangle 5$, $\langle 1 \rangle 6$ and induction on instructions evaluation.