N^{o} d'ordre : 3966 ANNÉE 2009

THÈSE / UNIVERSITÉ DE RENNES 1
sous le sceau de l'Université Européenne de Bretagne

pour le grade de

DOCTEUR DE L'UNIVERSITÉ DE RENNES 1

Mention : Informatique

Ecole doctorale Matisse

présentée par

Jérémy DUBREIL

préparée à l'unité de recherche IRISA (UMR 6074)
Institut de Recherche en Informatique et Systèmes Aléatoires
Composante universitaire : IFSIC

Monitoring and Supervisory Control for Opacity Properties

Thèse soutenue à Rennes le 25 novembre 2009

Remerciements

Je tiens à remercier Yassine Lakhnech et Jean-François Raskin pour avoir rapporté mon travail de thèse. Je remercie également Roland Groz pour avoir présidé ma soutenance de thèse ainsi que Philippe Darondeau et Olivier H. Roux pour avoir examiné pour travail.

Je remercie aussi chaleureusement mes encadrants de thèse, Thierry Jéron et Hervé Marchand pour leur disponibilité et leurs précieux conseils, distillées avec gentillesse et bonne humeur au court de mon doctorat. Grand merci de nouveau à Philippe Darondeau qui, au fil de nos collaborations, a beaucoup influencé mon travail de thèse et plus généralement ma vision du métier de chercheur.

Je voudrai remercier également l'ensemble de l'équipe VerTeCs. Il s'agit en effet d'une équipe où l'ambiance est particulièrement bonne et au sein de laquelle j'ai apprécié travailler pendant ces années de thèse.

Contents

Résumé des Travaux de Thèse

Nota bene: *This an extended summary of the thesis. This part in french is mandatory because the rest of the thesis is written in English.*

Le développement des réseaux ouverts tels qu'Internet ou les réseaux mobiles a induit l'explosion du nombre de service proposés sur ces réseaux. Certain de ces services manipulent des informations critiques qui doivent pas être corrompues de façon intentionnelle ou arriver en possession d'entités malveillantes. Citons pour exemple les systèmes d'administration électroniques, les systèmes de vote ou les bases de données d'information médicales. Dans ce contexte, le développement de techniques fiables et efficaces pour certifier la sécurité d'un système est essentiel. Afin d'étudier de tels algorithmes de certification, les propriétés de sécurité sont généralement classifiées en trois catégories :

- *l'intégrité* ;
- *la disponibilité* ;
- *la confidentialité*.

Une *politique de sécurité* consiste en un ensemble de propriétés de sécurité, de différentes catégories, qui doivent être conjointement satisfaites sur le système. Nous donnons quelques explications sur chacune de ces catégories afin de situer dans quel cadre se place nos travaux de thèse.

Les propriétés d'intégrité expriment l'idée qu'un attaquant ne peut exercer d'actions non autorisées ou forcer le système à atteindre une configuration critique. Si l'on choisit comme exemple un système de vote, le fait que personne ne puisse modifier le vote d'un autre électeur est une propriété d'intégrité. Les contraintes d'intégrité sont donc généralement exprimées par des propriétés de sûreté. Il existe néanmoins des propriétés d'intégrité qui ne s'expriment pas par des propriétés de sûreté, notamment lorsqu'il est question d'intégrité de l'information (voir [GMP92] pour plus de détails). Nous montrons dans cette thèse comment vérifier qu'une propriété de sûreté est satisfaite et comment assurer une telle propriété sur un système donné. Ces résultats s'appliqueront donc aux propriétés d'intégrité qui peuvent être exprimées par des propriétés de sûreté.

Les propriétés de disponibilité expriment l'idée qu'un attaquant ne peut entraver le bon comportement d'un système. En prenant de nouveau l'exemple d'un système de vote, un attaquant ne peut empêcher un électeur de voter. Typiquement, les attaques de type déni de service sont des violations de propriétés de disponibilité. Nous n'aborderons pas ici ce type de propriétés.

Les propriétés de confidentialité sont celles qui nous intéressent plus particulièrement dans cette thèse. Elles expriment l'idée qu'un attaquant ne peut acquérir d'information secrète. Par exemple, un attaquant ne peut inférer le vote d'un autre électeur. Dans ce document, nous allons considérer ces aspects de confidentialité avec la notion d'opacité. Cette notion a été introduite dans [Maz04] et ensuite généralisée au cas des systèmes de transitions dans [BKMR08].

La particularité des propriétés de confidentialité est qu'elles doivent être définies relativement à la connaissance des attaquants potentiels. Nous expliquons maintenant cet aspect avec un bref historique sur ce type de propriétés. Dans [BL73], les auteurs proposent une formalisation des systèmes alors en place dans le secteur militaire afin de préserver la confidentialité des informations. Ce modèle, connu sous les initiales BLP pour Bell et LaPadula, repose sur la notion d'objet (des documents par exemple) et de sujet (personnes ou programmes). Les sujets exercent des actions de type lecture, écriture, création, destruction, etc, sur ces objets. À chaque objet et chaque sujet est assigné un niveau de confidentialité, par exemple *confidentiel*, *publique*, etc. La confidentialité de l'ensemble est alors assuré par un contrôle d'accès qui interdit certain types d'opération. Par exemple, il est impossible pour un sujet de niveau *publique* de lire un objet situé au niveau *confidentiel*, ou encore, il est impossible pour un sujet de niveau *confidentiel* d'écrire dans un objet situé au niveau *publique*. Cet ensemble de règles a pour but d'empêcher le flot d'information du niveau *confidentiel* vers le niveau *publique*. Mais cette formalisation est limitée dans le sens où elle ne permet pas de réellement prouver l'absence de flot d'information. En effet, si un sujet A de niveau *publique* essaye d'écrire dans un fichier F au niveau *confidentiel* lorsque ce fichier est inexistant, alors A observe un message d'erreur. Si ce fichier F existe, l'écriture étant autorisé dans ce sens, A n'observe aucun message d'erreur. Ainsi, en collaboration avec un sujet B situé au niveau *confidentiel*, pour qui la création et la destruction d'objet est possible, le sujet B peut créer un canal de communication allant du niveau *confidentiel* au niveau *publique*, contournant ainsi les mesures de protection.

Le modèle BLP se révèle donc insuffisant pour interdire certains flots d'information car il ne permet pas de prendre en compte la capacité des sujets, potentiellement des attaquants, à inférer de l'information en fonction de ce qu'ils observent et de ce qu'ils connaissent du système. Pour palier à ce manque, Goguen et Meseguer proposent dans [GM82] une notion plus précise, appelé *non-interférence*, exprimant l'absence de flot d'informations confidentielles. Reprenant les dénominations utilisées plus haut, la propriété de non-interférence est vérifiée si ce que font les sujets de niveau *confidentiel* n'a pas d'influence sur ce que les sujets de niveau *publique* peuvent observer. On voit bien qu'avec une telle définition, le cas de flot d'information exprimé plus haut disparaît. Cette notion a ensuite été l'objet de nombreux travaux depuis comme par exemple [FG93, RS99, FG01] dont le but est de proposer différentes notions de non-interférence en utilisant le formalisme d'algèbre de processus CSP. Dans ces travaux, chaque notion dépend des hypothèses qui sont faite sur la capacité d'observation de l'attaquant et le type d'information secrète.

Dans [BKMR08], les auteurs étendent aux systèmes de transitions la propriété d'opacité introduite dans [Maz04] dans le cadre des protocoles cryptographique modélisés par des systèmmes de réecriture. Ils montrent alors que l'opacité est une propriété suffisamment générale pour pouvoir exprimer un ensemble non-négligeable d'autres propriétés de confidentialité telles que la non-interférence (SNNI) ou encore l'anonymat [SS96]. Cette notion d'opacité est le point de départ de ce travail de thèse et est définie par rapport à un attaquant qui a une pleine connaissance de la structure du système et qui en observe partiellement les exécutions. Nous définissons maintenant la propriété d'opacité.

Définition de l'opacité

Considérons un alphabet d'événements Λ et un ensemble d'états S. Ces ensembles Λ et S peuvent être infinis. L'ensemble des exécutions de la forme $s_0 \xrightarrow{\lambda_1} s_1 \ldots s_{n-1} \xrightarrow{\lambda_n} s_n$ qui peuvent être construites à partir de Λ et S en alternant les états et les événements est noté $E(\Lambda, S) = S(\Lambda S)^*$. Considérons un système critique modélisé par le LTS $M = (\Lambda, S, \delta, S_0)$ où S_0 dénote les états initiaux et $\delta : \Lambda \times S \to \mathcal{P}(S)$ est la fonction de transition. L'ensemble des exécutions possibles de M est noté $\mathcal{R}(M) \subseteq E(\Lambda, S)$ et le langage généré par M est noté $\mathcal{L}(M) = tr(\mathcal{R}(M))$ où tr est l'opérateur qui donne la trace d'une exécution, c'est à dire la séquence d'événements apparaissant dans cette exécution.

L'information secrète est donnée par un prédicat ϕ défini sur l'ensemble $E(\Lambda, S)$. Plus précisément, l'occurrence d'un run de M qui satisfait le prédicat ϕ constitue l'information qu'un attaquant ne doit pas pouvoir inférer. Considérons maintenant que l'observation de l'attaquant est définie par une fonction $obs : E(\Lambda, S) \to \mathcal{O}$ où $\mathcal{O}$ est l'ensemble des observations possibles. L'architecture que nous considérons est représentée sur la figure 0.1

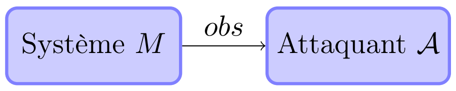


Fig. 0.1: Architecture Générale pour l'opacité

On dit alors que le système M est ϕ-opaque pour obs si pour toute exécution de M qui satisfait ϕ, il existe une autre exécution donnant la même observation et qui ne satisfait pas le prédicat ϕ. En d'autre terme :

$$\forall r \in \mathcal{R}(M),\ r \models \phi \implies \exists r' \in obs^{-1}(obs(r)) \cap \mathcal{R}(M),\ r \not\models \phi \tag{0.1}$$

Dans ce cas, l'attaquant observant $obs(r)$ ne pouvant deviner si r ou r' a été exécuté, il ne peut inférer si le prédicat ϕ a été satisfait par celle réellement exécutée par M.

Example 0.1 *Considérons le LTS M représenté par la figure 0.2 avec $\Lambda = \{h, \tau, a, b\}$. L'attaquant n'observe pas les états, mais observe les événements a et b. Le prédicat ϕ est satisfait pour les runs qui contiennent l'événement h. Sur cet example, le seul run*

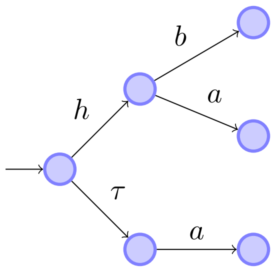


FIG. 0.2: Un exemple de non opacité

qui explique l'observation b contient l'événement h. L'attaquant peut donc inférer pour l'observation b que le prédicat ϕ est satisfait.

Nous allons nous intéresser dans la section suivante au problème de vérifier si un système est opaque.

Vérification de l'opacité

Nous faisons pour cette partie quelques hypothèses supplémentaires sur la fonction d'observation et sur le type de prédicat ϕ. Tout d'abord, nous supposons que l'attaquant observe un sous-ensemble Λ_a des événements de Λ. Nous définissons alors la projection $\pi_a : \Lambda^* \to \Lambda_a^*$ qui enlève d'un mot de Λ^* l'ensemble des événements qui n'appartiennent pas à Λ_a, c'est à dire, ceux qui ne sont pas observable par l'attaquant. La fonction d'observation est alors donnée par la fonction $p_a : E(\Lambda, S) \to \Lambda_a^*$ définie par $p_a = \pi_a \circ tr$. Ensuite, nous considérons le cas d'un prédicat ϕ défini par l'accessibilité d'un ensemble d'état $F(\phi) \subseteq S$.

Un moniteur pour ϕ est une fonction $\Gamma_\phi : \Lambda_a^* \to \{true, ?\}$ qui détermine si la satisfaction de ϕ par l'exécution courante de M peut être déterminée de façon sûre. Ainsi, le moniteur Γ_ϕ doit être correct dans le sens où il ne donne pas de faux verdict. La notion de correction peut se formaliser de la façon suivante :

$$\forall r \in \mathcal{R}(M),\ \Gamma_\phi(p_a(r)) = true \implies r \models \phi \tag{0.2}$$

Si un moniteur Γ_ϕ est correct et s'il existe un run $r \in \mathcal{R}(M)$ tel que $\Gamma_\phi(p_a(r)) = true$, alors l'attaquant peut, sans connaître r, inférer à partir de $obs(r)$ que le prédicat est satisfait par r et ainsi inférer de l'information secrète.

On peut remarquer que le moniteur défini par $\mathcal{O} \to \{true, ?\}$, $\mu \mapsto ?$ est correct mais ne présente pas d'intérêt pour détecter si le prédicat ϕ est satisfait étant donnée une observation. Nous cherchons alors à construire des moniteurs qui soient aussi complets par rapport au prédicat. C'est à dire que ? est la seule valeur possible pour les runs qui ne violent pas l'opacité :

$$\forall r \in \mathcal{R}(M),\ r \not\models \phi \implies \Gamma_\phi(p_a(r)) = ? \tag{0.3}$$

Dans ce cas, le problème de vérification de l'opacité se ramène au calcul d'un moniteur qui soit à la fois correct et complet. En effet, si Γ_ϕ est à la fois correct et complet, alors en combinant les expressions (0.2) et (0.3), nous obtenons :

$$\forall r \in \mathcal{R}(M),\ \Gamma_\phi(p_a(r)) = true \iff \forall r' \in obs^{-1}(obs(r)) \cap \mathcal{R}(M),\ r' \models \phi \tag{0.4}$$

Nous allons voir dans la section suivante comment le calcul de tels moniteurs peut être effectué.

Construction de moniteur pour l'opacité

La procédure de construction de moniteur que nous proposons dans cette thèse est basée sur l'opération de déterminisation par sous-ensemble d'état. Cette opération s'appuie sur les opérateurs :

- $post_M : \mathcal{P}(\Lambda) \to \mathcal{P}(S) \to \mathcal{P}(S)$ qui donne pour chaque ensemble d'événements $B \subseteq \Lambda$ et chaque ensemble d'état $X \subseteq S$ l'ensemble $post_M(B)(X)$ des états accessibles à partir de X après l'occurrence d'un événement de B ;
- L'opérateur $reach_M : \mathcal{P}(\Lambda) \to \mathcal{P}(S) \to \mathcal{P}(S)$ qui donne pour chaque $B \subseteq \Lambda$ et chaque $X \subseteq S$ l'ensemble $reach_M(B)(X)$ des états accessibles à partir de X après l'occurrence d'un nombre arbitraire mais fini d'événements de B. En d'autres termes, l'opérateur $reach_M$ est défini par

$$reach_M(B)(X) = lfp(Z \mapsto X \cup post_M(B)(Z))$$

La déterminisation de M est alors définie par $det_a(M) = (\Lambda_a, \mathcal{P}(S), \Delta, X_0)$ où $X_0 = reach_M(\Lambda_{ua})(S_0)$ avec $\Lambda_{ua} = \Lambda \setminus \Lambda_a$ et où la fonction de transition Δ_a est définie par

$$\begin{array}{rrcl} \Delta_a : & \Lambda_a \times \mathcal{P}(S) & \rightarrow & \mathcal{P}(S) \\ & \sigma,\ X & \mapsto & reach_M(\Lambda_{ua}) \circ post_M(\{\sigma\})(X) \end{array} \tag{0.5}$$

À partir de cette définition, on défini alors le moniteur :

$$\begin{array}{rrcl} \Gamma_\phi : & \Lambda_a^* & \rightarrow & \{true, ?\} \\ & \nu & \mapsto & \begin{cases} true \text{ si } \Delta_a(\nu, X_0) \subseteq F(\phi) \\ ? \text{ sinon} \end{cases} \end{array} \tag{0.6}$$

On montre alors que le moniteur Γ_ϕ est à la fois correct et complet. En d'autres termes, détecter des vulnérabilités pour l'opacité se ramène à un calcul d'accessibilité dans $det_a(M)$:

$$\Gamma_\phi^{-1}(true) \cap p_a(\mathcal{R}(M)) = \mathcal{L}(det_a(M), X_0, \mathcal{P}(F(\phi)))$$

Verifier si M est ϕ-opaque est alors équivalent à vérifier si le langage $\mathcal{L}(det_a(M), X_0, \mathcal{P}(F(\phi)))$ est vide.

Ainsi, si le LTS M est fini, c'est à dire si Λ et S sont finis, alors, le calcul de $det_a(M)$ et Γ_ϕ est possible. On peut dans ce cas calculer de façon exacte l'ensemble des traces observées qui révèlent le secret ϕ et le problème de vérification de l'opacité est alors décidable. Plus précisément, nous prouvons que ce problème est PSPACE-complet. Pour cela, nous explicitons le lien qui existe entre le problème d'universalité du language et l'opacité. Étant donné un automate A sur l'alphabet Σ, on dit de cet automate qu'il est langage universel si son langage accepté est Σ^*. Ce problème est connu pour être PSPACE-complet [SM73]. Nous montrons qu'une procédure pour résoudre le problème d'unversalité peut être adapté pour résoudre le problème de vérification de l'opacité, ce qui montre ce dernier est PSPACE. De plus, nous montrons qu'une procédure pour vérifier l'opacité permet aussi de résoudre le problème d'universalité, ce qui montre que le problème est PSCPACE-hard et donc PSPACE-complet.

Dans le reste de cette section, nous allons nous intéresser au cas où Λ et S ne sont pas finis.

Détection de vulnérabilité

Nous proposons deux approches différentes pour détecter les cas de violation de l'opacité. La première consiste à utiliser des techniques d'interprétation abstraite, [CC77a, CC92a, CC92b], pour abstraire l'opérateur $reach_M$ et ainsi obtenir une approximation de $det_a(M)$. Nous détaillons alors comment calculer un moniteur correct mais pas nécessairement complet à partir de cette approximation. La seconde approche consiste à utiliser une approximation régulière de M et à appliquer la théorie du diagnostic développée dans [SSL+95, SLS+96, JMPC06].

Utilisation de techniques issue de l'interprétation abstraite

Dans le cas où Λ et S sont infinis, il y a deux obstacles au calcul de $det_a(M)$. Le premier est que le LTS $det_a(M)$ peut être à branchement infini. Le second est que le calcul de $reach_M(\Lambda_{ua})$, nécessaire pour calculer Δ_a suivant (0.5), est basé sur un calcul de point fixe dans le treillis $\mathcal{P}(S)$. Rien ne garantit à priori que ce calcul termine toujours en un nombre fini d'itérations.

Nous allons donc contourner ces deux problèmes en utilisant des approximations. Tout d'abord, pour obtenir un LTS à branchement fini, nous considérons une relation d'équivalence $\theta \subseteq \Lambda \times \Lambda$ telle que l'ensemble des classes d'équivalences $\Sigma^\sharp = \{\theta(\lambda) : \lambda \in \Lambda\}$ définie une partition finie de Λ. Cette partition est supposée respecter l'observabilité des événements, c'est à dire :

$$\forall \lambda \in \Lambda,\ \lambda \in \Lambda_a \iff \theta(\lambda) \subseteq \Lambda_a$$

On note alors $\Sigma^\sharp_a = \theta(\Lambda_a)$ et $\Sigma^\sharp_{ua} = \theta(\Lambda_{ua})$.

Afin d'approximer $det_a(M)$, il est nécessaire d'approximer l'operateur $post_M$. Nous considerons alors donnée une connexion de Galois

$$(\mathcal{P}(S), \subseteq) \underset{\alpha^\sharp}{\overset{\beta^\sharp}{\rightleftarrows}} (Q^\sharp, \sqsubseteq^\sharp)$$

où $(Q^\sharp, \sqsubseteq^\sharp)$ est un treillis de hauteur finie. À partir de là, on peut définir les approximations supérieures correctes $post^\sharp_M$ de $post_M$ et $reach^\sharp_M$ de $reach_M$. C'est à dire que pour tout $B \subseteq \Lambda$ et pour tout $X \subseteq S$,

$$reach_M(B)(X) \subseteq \gamma^\sharp \circ reach^\sharp_M(B) \circ \alpha^\sharp(X)$$

Ces approximations sont alors toujours calculables. Ceci nous permet donc de calculer le LTS $det^\sharp_a(M) = (\Sigma^\sharp_a, Q^\sharp, \Delta^\sharp_a, q^\sharp_0)$ définie par $q^\sharp_0 = \alpha^\sharp(X_0)$ et

$$\Delta^\sharp_a(\sigma, q) = reach^\sharp_M(\Sigma^\sharp_{ua}) \circ post^\sharp_M(\{\sigma\})(q')$$

Dans ce cas, on montre que $\Delta^\sharp_a$ est une approximation supérieure correcte de Δ_a. Suivant la construction donnée en (0.6), on définit alors le moniteur :

$$\begin{array}{rccl} \Gamma^\sharp_\phi : & {\Sigma^\sharp_a}^* & \rightarrow & \{true, ?\} \\ & \nu & \mapsto & \begin{cases} true \text{ si } \gamma^\sharp \circ \Delta^\sharp_a(\theta^*(\nu), q^\sharp_0) \subseteq F(\phi) \\ ? \text{ sinon} \end{cases} \end{array} \tag{0.7}$$

où la fonction $\theta^* : \Lambda^*_a \rightarrow {\Sigma^\sharp_a}^*$ est définie inductivement par $\theta^*(\epsilon) = \epsilon$ et $\theta^*(\nu\lambda) = \theta^*(\nu)\theta(\lambda)$.

Le moniteur $\Gamma^\sharp_\phi$ est calculable car $Q^\sharp$ est de hauteur finie. De plus, ce moniteur est correct, c'est à dire que pour $r \in \mathcal{R}(M)$, alors $\Gamma^\sharp_\phi(p_a(r)) = true$ implique que $r \models \phi$. Par contre, ce moniteur n'est pas nécessairement complet. Ce résultat permet néanmoins à un attaquant de déduire de l'information confidentielle en approximant les comportements de M. Ceci implique alors une méthode pour détecter des vulnérabilités de M dynamiquement, c'est à dire pendant l'exécution du système.

Étudions maintenant comment vérifier si M est ϕ-opaque pour p_a et ceci statiquement, c'est à dire sans exécuter le système. Il y a deux aspects qui dépendent des caractéristiques de la propriété d'opacité qui nous empêchent d'utiliser directement le moniteur $\Gamma^\sharp_\phi$ pour certifier ou invalider statiquement l'opacité de M.

Premièrement, même dans le cas où le treillis $Q^\sharp$ est fini, le moniteur $\Gamma^\sharp_\phi$ n'étant pas complet, il se peut que ${\Gamma^\sharp_\phi}^{-1}(true) \cap p_a(\mathcal{R}(M)) = \emptyset$ sans que le système ne soit opaque. En d'autres termes, le fait de considérer une approximation supérieure fait perdre de la précision quand à l'ensemble des exécutions compatibles avec une observation. Ainsi il se peut que le verdict ? soit dû à des exécutions ne satisfaisait pas ϕ qui sont possibles d'après $det^\sharp_a(M)$ mais qui ne n'appartiennent pas à $\mathcal{R}(M)$.

Deuxièmement, l'ensemble des comportements de M étant inconnu à priori, il est n'est pas toujours possible de décider si ${\Gamma^\sharp_\phi}^{-1}(true) \cap p_a(\mathcal{R}(M)) \neq \emptyset$ et donc d'invalider l'opacité de cette façon. Il nous faut alors utiliser une sous-approximation de l'ensemble $p_a(\mathcal{R}(M))$ pour pouvoir exhiber une trace observée $\nu \in p_a(\mathcal{R}(M))$ telle que $\Gamma^\sharp_\phi(\nu) = true$. Dans cette optique, nous supposons que $|Q^\sharp| < \infty$ et considérons un treillis fini $Q^\flat$ ainsi qu'une connexion de Galois

$$(Q^\flat, \sqsubseteq^\flat) \underset{\alpha^\flat}{\overset{\beta^\flat}{\leftrightarrows}} (\mathcal{P}(S), \subseteq)$$

permettant de sous-approximer les ensembles d'états. Nous considérons une partie finie $\Sigma^\flat \subseteq \Lambda$ de l'ensemble de événements de M, avec $\Sigma^\flat_a = \Lambda_a \cap \Sigma^\flat$ et $\Sigma^\flat_{ua} = \Lambda_{ua} \cap \Sigma^\flat$. À partir de cette connexion de Galois et de cet ensemble fini d'événements, nous calculons une sous-approximation inférieure correcte $det^\flat_a(M)$ de $det_a(M)$ qui est un LTS fini tel que $\mathcal{L}(det^\flat_a(M)) \subseteq p_a(\mathcal{R}(M)) \cap {\Sigma^\flat_a}^*$. Ainsi, nous pouvons faire le lien avec la sur-approximation utilisée plus haut en utilisant le résultat :

$$\forall r \in E(\Lambda, S),\ p_a(r) \in {\Gamma^\sharp_M}^{-1}(true) \cap \mathcal{L}(det^\flat_a(M)) \implies r \in \mathcal{R}(M) \wedge r \models \phi$$

Ainsi, le fait que ${\Gamma^\sharp_M}^{-1}(true) \cap \mathcal{L}(det^\flat_a(M)) \neq \emptyset$ implique que le système M n'est pas ϕ-opaque pour p_a.

Ces travaux sont présentés en section 4.3 et ont été publiés en [Dub09].

Application du diagnostic avec des abstractions régulières

Nous considérons maintenant que l'alphabet d'événements de M est fini et nous notons alors Σ cet ensemble d'événements. Nous ne faisons pas d'hypothèse sur l'ensemble des états qui peut être infini. Ainsi, le langage de M n'est pas nécessairement régulier. Dans ce contexte, nous supposons que le prédicat ϕ est défini par rapport à la trace générée par une exécution. C'est à dire qu'il existe un langage $L(\phi) \subseteq \Lambda^*$ tel que $r \models \phi$ si $tr(r) \in L(\phi)$. Nous supposons aussi que ce langage $L(\phi)$ est régulier. Nous montrons que l'on peut considérer le problème de vérification de l'opacité avec une approche basée sur les langages. Ainsi, l'objectif est de détecter si certain mots de $\mathcal{L}(M)$ révèlent le secret.

L'approche présentée ici consiste à considérer une approximation régulière de M, c'est à dire un LTS fini G tel $\mathcal{L}(M) \subseteq \mathcal{L}(G)$. Dans ce cas, G étant fini, on peut calculer un moniteur correcte pour détecter les cas de non-opacité à partir des traces observées de M. Un attaquant peut alors inférer la satisfaction de ϕ à partir de G en s'appuyant sur le résultat suivant :

$$\forall \mu \in \pi_a(\mathcal{L}(M)),\ \emptyset \subsetneq \pi_a^{-1}(\mu) \cap \mathcal{L}(G) \subseteq L(\phi) \implies \emptyset \subsetneq \pi_a^{-1}(\mu) \cap \mathcal{L}(M) \subseteq L(\phi)$$

Par contre, le résultat précédent ne peut pas être utilisé pour vérifier statiquement l'opacité de M pour les mêmes raisons que dans le cas d'approximations basées sur des techniques d'interprétation abstraite. Nous présentons alors une technique basée sur la théorie du diagnostic pour détecter en ligne l'occurrence des fuites d'informations. Nous étudions alors le problème de définir et calculer un diagnostiqueur D dont le but est de détecter sous observation partielle l'ensemble des occurrences de fuite d'information. Pour cela, nous supposons que le diagnostiqueur observe les événements de l'alphabet $\Sigma_o \subseteq \Sigma$. L'architecture que nous considérons est représenté par la figure 0.3.

Étant donnée une propriété de sûreté ψ définie par le langage clos par préfixe $L(\psi) \subseteq \Sigma^*$, un langage L est diagnosticable s'il existe une borne N telle que pour tout mot de L qui ne satisfait pas ψ, il suffit d'attendre l'occurrence d'au plus N événements pour pouvoir, sous

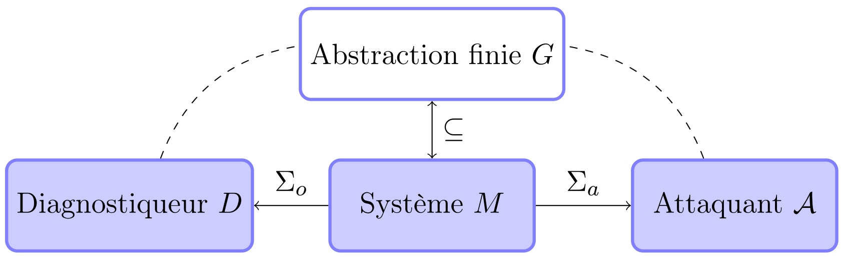


FIG. 0.3: Détections de vulnérabilités utilisant diagnostique et abstraction régulière

observation partielle, inférer de manière sûre le prédicat ψ n'est pas satisfait. Formellement :

$$\forall w \in L \cap L(\neg\psi),\ \forall w' \in w^{-1}L,\ |w| \geq N \implies \pi_o^{-1}(\pi_o(ww')) \cap L \subseteq L(\neg\psi)$$

Nous donnons ensuite une procédure pour vérifier la diagnosticabilité pour langage régulier L et un prédicat ψ donné aussi par un langage régulier $L(\psi)$. Nous montrons aussi que la diagnosticabilité est préservé par l'inclusion, ce qui va nous permettre de conserver cette propriété sur $\mathcal{L}(M)$ lorsqu'elle est vérifié sur l'abstraction $\mathcal{L}(G)$ de $\mathcal{L}(M)$.

À partir de l'abstraction G, nous pouvons alors définir le prédicat ψ par :

$$L(\psi) = \mathcal{L}(M) \setminus (\{w \in \mathcal{L}(G) : \pi_a^{-1}(\pi_a(w)) \subseteq L(\phi)\}\ \Sigma^*)$$

The langage $L(\psi)$ est l'ensemble des sequences de $\mathcal{L}(M)$ pour lesquelles le secret ϕ n'a pas été révélé à l'attaquant utilisant un moniteur correct calculé à partir de G. Alors, le résultat principal de cette section est : si le langage $\mathcal{L}(G)$ est diagnosticable pour ψ pour une borne N, alors tous les cas de fuite d'information vont être détectées au plus N occurrences après qu'elles se soient produites dans $\mathcal{L}(M)$.

Ces travaux sont présentés dans la section 4.4 et font suite aux publications [DJM07] et [DJM09].

Assurer l'opacité sur un Système

Dans cette partie, nous étudions deux approches pour garantir la propriété d'opacité sur le système M, supposé fini. Dans ce cas, on note Σ pour l'alphabet d'événements et Q sur l'ensemble d'états, notant ainsi $M = (\Sigma, Q, \delta, Q_0)$.

La première approche pour assurer l'opacité de M consiste à restreindre les comportements du système à un sous ensemble de sorte que l'opacité soit vérifiée sur ce sous-ensemble. Pour cela, nous utilisons la théorie du contrôle à la Ramadge et Wonham [RW87, RW89] qui consiste à calculer un contrôleur C tel que la composition parallèle $C \parallel M$ soit opaque. La seconde approche consiste à modifier dynamiquement l'observabilité des événements de M de façon à limiter la capacité de déduction de l'attaquant et préserver le secret ϕ.

Synthèse de contrôleur pour assurer l'opacité

Nous supposons ici que l'attaquant observe les événements d'un sous-alphabet $\Lambda_a \subseteq \Lambda$, et le prédicat ϕ est donné par un langage régulier $L(\phi)$. On peut alors suivre une approche s'appuyant sur les langages et considérer la projection $\pi_a : \Lambda^* \to \Lambda_a^*$. Dans ce cas, nous supposons aussi que M est déterministe, c'est à dire $M = (\Sigma, Q, \delta, q_0)$ avec δ qui est une fonction partielle de $\Sigma \times Q$ dans Q.

L'objectif est ici de calculer un contrôleur C tel que le langage $\mathcal{L}(C \parallel M)$ soit ϕ-opaque pour π_a. Ce contrôleur doit obéir à quelques contraintes. La première de ces contraintes est que le contrôle s'exerce sous observation partielle, c'est à dire que le contrôleur n'observe que les événements de l'alphabet $\Sigma_o \subseteq \Sigma$. Le contrôleur doit alors être tel que $\mathcal{L}(C) \subseteq \Sigma_o^*$. Nous supposons aussi que tous les événements de Σ_o ne peuvent être empêchés par contrôle. De plus, nous cherchons un contrôleur qui soit le plus permissif possible, c'est à dire qu'il ne restreint pas inutilement le comportement de M.

Nous appelons un *langage contrôlé* un sous langage de $\mathcal{L}(M)$ qui soit non-vide, clos par préfixe, *contrôlable* et *normal*. Un sous-langage contrôlé est un langage qui peut être obtenu par la composition $C \parallel M$ pour un contrôleur C observant les événements de Σ_o et ne bloquant que l'occurrence des événements de Σ_c. Le problème peut donc être reformulé de façon équivalente au calcul, lorsque celui-ci existe, d'un langage contrôlé K qui soit maximal au sens de l'inclusion des langages.

Nous proposons une solution à ce problème lorsque les alphabets Σ_a et Σ_o sont comparables, c'est à dire lorsque $\Sigma_a \subseteq \Sigma_o$ ou $\Sigma_o \subseteq \Sigma_a$.

Dans un premier temps, nous étudions l'application d'une technique classique pour la synthèse de contrôleur. Cette technique, dite de Ramadge et Wonham consiste à appliquer alternativement l'opérateur Op, qui associe à un langage L le plus grand sous-langage clos par préfixe de L qui soit ϕ-opaque pour π_a, et l'opérateur CN, qui associe à un langage L son plus grand sous sous-langage clos par préfixe, normal et contrôlable. L'idée de l'algorithme est le suivant :

- Partant de $\mathcal{L}(M)$, le langage $L_1 = Op(\mathcal{L}(M))$ est opaque et clos par préfixe mais n'est pas nécessairement normal et contrôlable.
- On applique alors à L_1 l'opérateur CN pour obtenir $K_1 = CN(L_1) = CN \circ Op(\mathcal{L}(M))$. Le langage K_1 est normal et contrôlable mais l'opacité n'est pas nécessairement préservé par l'opération.
- L'opérateur $Op \circ CN$ est monotone dans le treillis complet des sous-langages préfixe-clos de $\mathcal{L}(M)$ et admet donc un plus grand point fixe.

– Ce point fixe $gfp(Op \circ CN)$ est le plus grand sous-langage contrôlé qui soit ϕ-opaque pour π_a.

Mais rien de garantie que ce point fixe termine toujours ni que le langage obtenu soit non-vide et régulier. Nous présentons donc quelques conditions suffisantes pour que ce calcul de point-fixe termine après un nombre fini d'itérations, ce qui implique aussi que le langage obtenu est régulier car les opérateurs Op et CN préservent la régularité des langages. Nous montrons que le calcul de point fixe termine lorsque $\Sigma_o \subseteq \Sigma_a$ et $\Sigma_a \subseteq \Sigma_c$.

Example 0.2 *Afin d'illustrer l'algorithme présenté ci-dessus, considérons le LTS M donné par la figure 0.4. Nous supposons que $\Sigma_a = \{a, b, d, e\}$, $\Sigma_o = \{a, c_1, c_2, b, d, e\}$, (donc seul l'événement τ est inobservable par le contrôleur) et $\Sigma_c = \{b, c_1, c_2, e\}$. Le prédicat ϕ est défini par le langage régulier $L(\phi) = \Sigma^* h \Sigma^*$. En observant d, l'attaquant est sûr que l'événement h s'est produit et le secret est alors révélé. L'opérateur Op enlève donc du langage du M l'événement d et tout ses suffixes, c'est à dire l'ensemble des mots donc le suffixe appartient à da^*. Mais le language obtenu n'est pas contrôlable. Appliquer l'opérateur CN revient à bloquer l'action c_1 à l'état (1). Le LTS obtenu est donné par la figure 0.5(a). Cependant, le secret peut encore être révélé à l'attaquant sur le langage obtenu, notamment, si l'attaquant observe b. Ceci conduit à couper l'événement c_2 à l'état (7). Le LTS obtenu, représenté sur la figure 0.5(b) est à la fois opaque, contrôlable et normal. Nous avons alors obtenu le plus grand langage controllé assurant la propriété d'opacité sur M.*

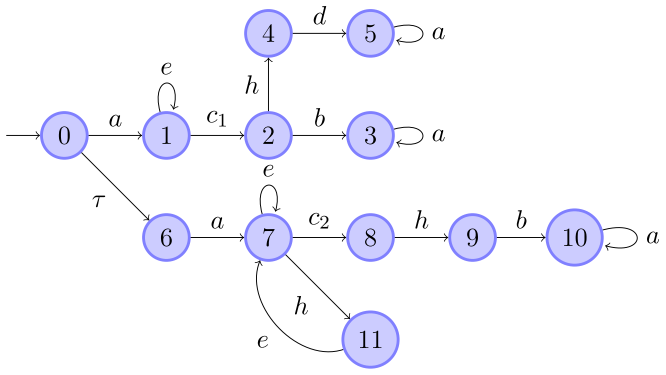
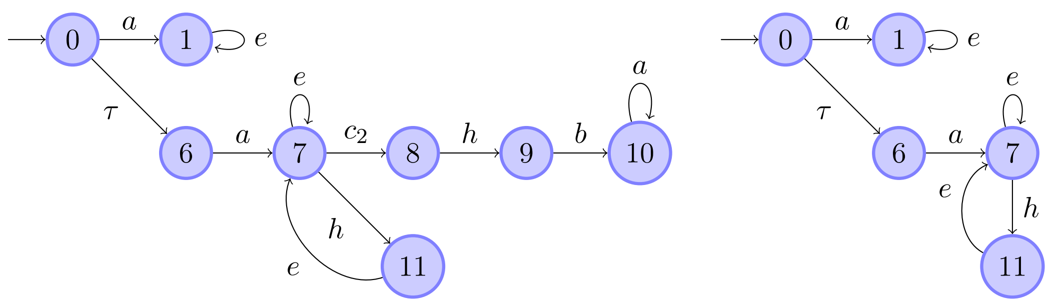


FIG. 0.4: Assurer l'opacité par contrôle (I)

Nous donnons aussi dans cette thèse un contre-exemple, dans le cas $\Sigma_a \subset \Sigma_o$, pour montrer que le calcul de point fixe décrit plus haut ne termine pas toujours. Sous l'hypothèse $\Sigma_a \subset \Sigma_o$, nous présentons alors une autre technique qui prends en compte plus précisément les spécificités de la propriété d'opacité dans le calcul du langage contrôlé. Premièrement, nous remarquons que l'on peut faire l'hypothèse $\Sigma_o = \Sigma$ sans perdre de généralité. En effet, étant donnée une solution pour le problème de contrôle à partir du langage projeté



(a) Step 1 (b) Step 2

Fig. 0.5: Assurer l'opacité par contrôle (II)

$\pi_o(\mathcal{L}(M))$, nous montrons comment retrouver le résultat pour le problème de contrôle initial dans le cas $\Sigma_o \subseteq \Sigma$. Sous cette hypothèse $\Sigma_o = \Sigma$, le problème de calculer le plus grand sous-langage contrôlé qui soit opaque peut se voir de façon équivalente comme le calcul d'une fonction de contrôle $f : \Sigma^* \to \mathcal{P}(\Sigma)$ maximal au sens de l'inclusion point à point et telle que pour tout $w \in \Sigma^*$, $\Sigma_{uc} \subseteq f(w)$ (avec $\Sigma_{uc} = \Sigma \setminus \Sigma_c$). Nous montrons alors que cette fonction de contrôle ne dépend que de l'état de M et de l'estimé d'état de l'attaquant, c'est à dire l'ensemble des états possiblement atteints compte-tenu de la trace observée. Cette configuration constitué de l'estimé de l'attaquant et de l'état du système est donnée par la fonction $\zeta : \Sigma^* \to \mathcal{P}(Q) \times Q$. Il existe alors une fonction $\bar{f} : \mathcal{P}(Q) \times Q \to Q$ telle que $f = \bar{f} \circ \zeta$. Ce résultat implique que le plus grand sous-langage contrôlé assurant l'opacité est régulier et induit aussi un algorithme pour son calcul effectif. En effet, nous remarquons que si $(e, q) = \zeta(w)$, alors $\bar{f}(e, q) \subseteq \delta(\cdot, q)^{-1}(Q)$. En d'autre terme, la fonction $\bar{f}$ induit une restriction de δ paramétrée par l'estimé e de l'attaquant. L'algorithme que nous proposons consiste alors à considérer toutes les restrictions possibles de δ pour chaque estimé $e \subseteq Q$. Ceci revient à considérer l'ensemble des fonctions $d : \mathcal{P}(Q) \times \Sigma \times Q$ telle que pour tout $e \subseteq Q$, la fonction $d(e, \cdot, \cdot)$ est définie et égal à δ seulement lorsque cette dernière est définie. Chaque fonction d définie un sous langage de M sur lequel on peut calculer les mots qui révèlent de l'information secrète. Partant de la fonction d_0 définie par $d_0(e, \cdot, \cdot) = \delta$, on défini alors la suite définie par $d_{i+1} = \alpha(d_i)$ où la fonction α enlève d'une fonction d l'ensemble des transitions qui conduisent à une fuite d'information. Cette suite converge avec un nombre fini d'itérations vers une fonction d_N, et nous montrons alors que $d_N = \bar{f}$. Ainsi, connaissant la valeur de $\bar{f}$ pour chaque configuration $(e, q) \in \mathcal{P}(Q) \times Q$, nous en déduisons la valeur de f et par conséquent le plus grand sous-langage contrôlé $K \subseteq \mathcal{L}(M)$ tel que K soit ϕ-opaque pour π_a.

Ces travaux sont présentés au chapitre 5 et font suite aux publications [DDM08] et [DDM09].

Masquage d'information par des projections dynamiques

Nous présentons dans cette section une technique qui consiste à modifier dynamiquement l'observabilité des événements afin de préserver la propriété d'opacité. Cette technique a été introduite dans [CT08] dans le cadre du diagnostique. Ces travaux sont présentés dans le chapitre 6 et fait suite à la publication de [CDM09a]. Le problème est de trouver une projection dynamique π_T assurant l'opacité d'un système fini $M = (\Sigma, Q, \delta, Q_0)$, possiblement non-déterministe par rapport à un prédicat ϕ défini par l'ensemble d'état $F(\phi) \subseteq Q$.

Pour définir une telle projection, nous supposons que l'ensemble des événements observables Σ_a est partitionné entre les événements Σ_v qui peuvent être masqués si besoin, par exemple les sorties du dystème, et les événements Σ_{uv} qui ne peuvent jamais être masqués, par exemple les entrées de l'attaquant. Nous considérons ici que le choix de masquer ou non un événement dépend de la trace observée par l'attaquant. Dans ce contexte, la notion de choix d'observation est une fonction $T : \Sigma_a^* \to \mathcal{P}(\Sigma_a)$ telle que pour tout $\mu \in \Sigma_a^*$, $\Sigma_{uv} \subseteq T(\mu)$[1]. Cette notion de choix dynamique nous permet de définir celle de projection dynamique de façon inductive : étant donné un choix d'observation T, on défini la projection π_T par :

$$\begin{array}{rrcl} \pi_T : & \Sigma^* & \to & \Sigma_a^* \\ & \epsilon & \mapsto & \epsilon \\ & w\sigma & \mapsto & \begin{cases} \pi_T(w)\sigma \text{ si } \sigma \in T(\pi_T(w)) \\ \pi_T(w) \text{ sinon} \end{cases} \end{array}$$

Nous disons d'un choix d'observabilité qu'il est valide s'il défini une projection dynamique qui assure l'opacité de ϕ sur M.

Afin de préserver au maximum le service fournit par le système, qui peut s'exprimer comme une propriété sur les traces observées, il peut être intéressant de rechercher une projection dynamique qui masque le moins possible d'événements de Σ_v. Malheureusement, l'ensemble des choix d'observabilité valides n'est pas clos par union, ce qui implique qu'il n'existe pas forcément un unique choix d'observabilité maximal par rapport à la relation d'inclusion. Néanmoins, nous montrons que cet ensemble de choix dynamiques valides peut être représenté de façon finie par l'arène (finie) d'un jeu de sûreté à deux joueurs. Pour cela, nous définissons le jeu alternant suivant :

– Les actions du joueur 1 appartiennent à l'ensemble $\Upsilon_1 = \{t \subseteq \Sigma_a : \Sigma_{uv} \subseteq t\}$ et les états pour lesquels le joueur 1 a la main est $S_1 = \mathcal{P}(Q)$;
– Les actions du joueur 2 appartiennent à l'ensemble $\Upsilon_2 = \Sigma_a$ et ses états sont de la forme $S_2 = S_1 \times \Upsilon_1$;
– l'arène du jeu est défini par le LTS fini $H = (\Upsilon_1 \cup \Upsilon_2, S_1 \cup S_2, \delta_H, Q_0)$ où $Q_0 \in S_1$ est

[1]Cette contrainte est similaire à la notion de non-contrôlabilité.

l'état initial du jeu ;
- la fonction de transition δ_H induit le déroulement du jeu et est définie par :
  - pour $e \in S_1$ et $t \in \Upsilon_1$, $\delta_H(t, e) = (e, t) \in S_2$;
  - pour $(e, t) \in S_2$ et $\sigma \in \Upsilon_2$, si $\sigma \in t$ et $e' = post_M(\{\sigma\}) \circ reach_M(\Sigma \setminus t)(e) \neq \emptyset$ alors $\delta_H(\sigma, (e, t)) = e' \in S_1$ et est indéfini dans les autres cas.

L'objectif du jeu est donné par rapport à l'ensemble d'état

$$Bad = \{(e, t) \in S_2 : reach_M(\Sigma \setminus t)(e) \subseteq F(\phi)\}$$

Le joueur 2 cherche à atteindre un état de *Bad* et joue ainsi un jeu d'accessibilité. Le joueur 1 cherche de son coté à éviter que le joueur 2 gagne, c'est à dire qu'il cherche à jouer de façon à ce que les états de *Bad* ne soit pas atteints. Ce jeu de sûreté sur une arène finie est déterminé d'après [Mar75], c'est à dire que soit le joueur 1, soit le joueur 2 a une stratégie gagnante pour gagner (à partir de l'état initial).

À partir de ce jeu, nous montrons dans un premier temps qu'il existe une correspondance bijective entre l'ensemble des choix d'observabilité et l'ensemble des stratégies du joueur 1. Dans un second temps, nous montrons qu'un choix d'observabilité est valide si et seulement si il définit une stratégie gagnante du joueur 1. Dans un troisième temps, nous montrons que l'ensemble des stratégies gagnante du joueur 1 peut être calculé avec une complexité polynomial en la taille de H (qui a lui une taille exponentielle en la taille de M).

Il existe donc une projection dynamique assurant l'opacité de ϕ sur M si et seulement si il existe une stratégie gagnante pour le joueur 1. L'ensemble des choix d'observabilité valides est représenté par l'arène $\mathcal{H} = (\Upsilon_1 \cup \Upsilon_2, S_1 \cup S_2, \delta_{\mathcal{H}}, Q_0)$ où $\delta_{\mathcal{H}}$ est obtenu à partir de δ_H en éliminant toutes les actions du joueur 1 qui permettent au joueur 2 de gagner le jeu, c'est à dire atteindre un état de *Bad*.

Ces travaux sont présentés au chapitre 6 et font suite à la publication de [CDM09a].

Conclusion

Dans cette thèse, nous présentons des méthodes de vérification de l'opacité basées sur des techniques d'interprétation abstraite et sur l'application du diagnostique. Nous présentons aussi des algorithmes de construction de système opaque qui s'appuient soit sur des techniques de synthèse de contrôleur soit sur la notion de projection dynamique.

Nos travaux actuels portent premièrement sur une implémentation des techniques de vérification de l'opacité et de détection de vulnérabilités. Un prototype implémentant les calculs de moniteur et de vérification de la diagnosticabilité dans le cas fini a été développé. L'objectif est alors de fusionner les approches présenter en section 4.3 et 4.4 afin de pouvoir utiliser des techniques d'interprétation abstraite pour calculer des moniteurs dans le cas de systèmes infinis.

Aussi, les travaux présentés aux chapitres 5 et 6 présentent de forte similitudes à la fois dans la formalisation des deux problèmes et dans les solutions algorithmiques présentées. Il serait donc intéressant de pouvoir les fusionner en un unique problème afin de voir si les techniques de synthèse de contrôleur peuvent permettre d'étendre les techniques de calcul de projections dynamiques à des choix d'observabilité qui dépendent des mots générés par M et non plus seulement des traces observées par l'attaquant.

1 Introduction

Ensuring the confidentiality of critical information manipulated by computer systems has become one of the most challenging objectives of modern hardware and software design. Interconnected networks, like Internet or mobile phones, providing communication services, decision taking facilities or access to information, are open by nature and therefore vulnerable to malicious attackers. Moreover, the kind of services proposed through those networks has changed during the past decade which has seen the emergence of services like Internet banking, e-government, e-voting systems, medical information storage and even in recent years remote surgery. Such services handle critical information that should neither be corrupted nor leaked to unauthorized users. In practice, the level of security of an information system is often determined by the quantity of known and public vulnerabilities. This approach, possibly forgetting vulnerabilities that are known only by a malicious group of users, is not satisfying regarding the significant place of information technologies in such critical sectors like medicine, e-government or finance. For example, large scale stealing of medical records or massive modification of votes on e-voting systems can have dramatic consequences. Then, there should be no security breaches on such infrastructures. Furthermore, real security of systems is not sufficient alone. Indeed, even if a system happens to be secure, it is also essential for the users to know that it is secure. In other words, the guarantees about the security of a critical system can be part of the services it provides. For example, an e-voting system based on Internet can be successfully deployed only if the electors can trust that the system will not allow a particular candidate to influence the outcome of an election. Therefore, an independent third party must be able to prove that there cannot be frauds in the elections. This situation implies the development of reliable methods for certifying the absence of security breaches on such critical systems. Unfortunately, manual analysis can be very expensive, permeable to mistakes and require a high level of expertise. Moreover, a manual analysis is often impossible to achieve in practice for large infrastructures, especially when updates are regularly performed.

In this context, beside the large amount of work that has been done for several decades in the domain of cryptography, see [MVO96], there has been a growing interest in the application of formal methods to verify security properties. To cite only few of the works in this domain, in [Low99] the author applies model-checking techniques to verify cryptographic protocols. This permitted to discover some flaws in some widely deployed and long standing security protocols. In [AG99, BAF05], the authors develop a process algebra framework such that some security properties, mostly information flow properties, are verifiable. There have been also important contributions to the formal analysis and enforcement of security properties based on monitoring techniques. Some notable works in this area have been published in [Sch00] and later extend in [LBW05]. In these articles, the authors consider automata that can stop or modify at runtime the behavior of a program to enforce a security policy, mostly consisting in safety properties.

In order to automate the certification process, security requirements have been formalized. The security properties that a critical system shall satisfy are generally classified into three categories (see [Bis04] for a complete review):

- *integrity*;
- *availability*;
- *confidentiality.*

We will consider an e-voting system to give some examples about each category of property.

Integrity properties express that users cannot perform some actions they have not been allowed to. For example, requiring from an e-voting system that votes cannot be modified or deleted by a third party is a concern of integrity. In practice, integrity properties are often given as safety properties, like for example "malicious users cannot remove protected files". But the term integrity can sometimes be used for integrity of information and then not always expressible as safety properties. For example, consider that Alice wants to send the message m to Bob using a flawed cryptographic protocol. Then, an attacker eavesdropping the message m and sending the message m', $m' \neq m$ to Bob will not violate any safety property. Such integrity property can be formulated in terms of knowledge [GMP92]: "If Alice knows that the message is m before the transmission (which is obvious in that case), then Bob must know that this message is m once the transmission has terminated."

Availability properties express that malicious users cannot disrupt the expected behavior of a service. For example, requiring that every elector can vote is a concern of availability. A Denial of Service attack is a malicious effort orchestrated to disrupt and make unavailable services such as online banks, credit card payment system as well as government or political websites. Such security issues are typical concerns of availability. Intuitively, some availability properties can be expressed by liveness. For example, if a user sends the correct credit card information, then an online payment system must proceed to the money transfer.

The last category of security properties is the confidentiality. It expresses that unauthorized users cannot acquire secret information. In an e-voting system, requiring that no third party can infer the vote of an elector is a concern of confidentiality. For example, consider the following voting scheme: Let M be a voting system where the values of the votes have to remain confidential. The order of the voters is random but observable whereas the values of the votes are not observable. Suppose that voting is stopped as soon as the outcome of the election is certain. Then, one can infer the vote of the last voter and such voting scheme is then insecure. Another scheme where the outcome of the election is published once every elector has voted does not present the same flaw.

Finally, in the remainder of the thesis, we will call a *security policy* a set of security properties that have to be simultaneously satisfied.

In opposition to a classical approach to ensure security by fixing exposed vulnerabilities, formal certification is especially crucial in the context of confidentiality properties. Clearly, if a service needs to keep some information secret, like the values of the votes or patient's data on an electronic health record system, then formal certification is essentially the only way to obtain confidentiality guarantees since information flow attacks may not let any sign that some secret information has leaked. Then, the occurrence of such attacks may be hard to detect and the damages intricate to recover.

In the thesis, we will mostly focus on confidentiality with the notion of *opacity* [BKMR08]. We will investigate some techniques to verify, monitor and enforce opacity properties. We will generally adapt to opacity some techniques that have been developed for safety properties. Then, to outline the specificities of opacity and to illustrate the techniques we propose for opacity related problems, we will also recall how such techniques can solve safety related ones. The presented results can therefore be applied to handle integrity properties expressible as safety properties. Finally, the case of availability properties is not investigated in this document.

We consider a critical system, whose behavior is modeled by a possibly infinite labeled transition system, required to keep secret some confidential information against inquisitive attackers. We assume that attackers are partially observing the executions of the system. For example, attackers may observe the interactions, i.e. the inputs and outputs sent to and received from the system. But we can also consider that timing information, power consumption, or electromagnetic radiations are observable as long as they can be modeled in an event based fashion. This partial observation is given as a function mapping the runs of the system to a set of observations. For security analysis, it is also common to consider that attackers may have a complete knowledge about the structure of the system. We will make this assumption throughout the thesis. This would be realistic when analyzing standardized protocols or software with sources publicly available. But, even in other cases, it is reasonable to assume that attackers can always apply retro-engineering techniques to acquire information about the ways a system works. Moreover, we will show that the most precise the knowledge of the attacker is, the most accurate its ability to infer information will be. Then, considering an attacker having full information about the behavior of the system is considering the worst situation from a security point of view. In other words, conditions for systems confidentiality stated in this context will also hold in the less restrictive cases where the attacker only has imprecise information about the system.

Confidentiality requirements express that unauthorized users, more simply called attackers, should not be able to access secret information. For example, if an operating system fails to protect a file storing the passwords, and users can directly open it, then the confidentiality of the password is broken. This situation can be expressed as a safety property but this formalization is generally too weak to really capture confidentiality requirements as the following example illustrates.

Example 1.1 *Consider the pseudo-program P:*

```
int k = 0
int x = 0

k = random()
x = random()

while true
  receive(m, A)
  if (m+1) != k then
    send(x+1, A)
  else
    send(0, A)
```

The value of `k` *is secret. Knowing the code of P, a malicious user A can follow the strategy:*

- *choose a value, for example 5;*
- *send the message 5 to P (with the event $receive(5, A)$);*
- *if the answer is 0, then A can deduce that the value of* `k` *is 6.*

The attacker cannot directly read the value of `k` *but can nevertheless infer this information in some favorable cases.*

In the situation of the previous example, we will say that there is an implicit flow of secret information, i.e. the attacker is able to infer secret information without directly getting it, but deducing it from what is observed. The notion of *opacity*, introduced in [Maz04] and adapted to transition systems in [BKMR08], formalizes the ability of a system to keep secret some critical information, assuming that attackers have a complete knowledge of the structure of the system and partially observing its executions. Given a predicate over the executions whose satisfaction is secret information, the predicate is opaque when for every observation, there exists a run that is possible regarding this observation and which does not satisfy the predicate. Then, an attacker observing the system cannot infer from this (partial) observation that the current run of the system belongs to the secret. In addition to the simplicity of its definition, the advantage of the notion of opacity is its expressiveness since other classical notions of information flow properties like anonymity or non-interference can be formalized as opacity problems [BKMR08].

To illustrate the notion of opacity, consider a program P to be an implementation of a cryptographic primitive that needs to keep secret the value of some key k. This value is randomly generated when P is initialized and is not modified during the execution of P. Let the predicate ϕ be defined as the set of executions of P that are possible only when the value of k is smaller than a given value v. If ϕ is not opaque on P, then for some observation ν, the attacker knows that the current execution of P belongs to ϕ, and then that $0 \leq k \leq v$ since every execution of P that can explain the observation ν belongs to ϕ. In this example, the program P also leaks information if the attacker can infer the truth of $\neg\phi$, as it implies that $v < k$.

For another illustration, let M be a basic authorization service. The users send their logins and passwords to M which reads from a database file `pwd` if the values match and decides then whether to let users proceed to the next step. The file `pwd` is normally protected, nobody can read its content, but in order to make the comparison, M enables read access on `pwd`, compares the values and then disables read access. The attacker wants to infer that `pwd` can be read to launch a `print pwd` command and steal all login information. Therefore, consider the predicate that is true on the runs whose last state is such that the password file is read accessible. If this predicate is opaque, then the attack explained above have no certainty to succeed. We can see on this example, inspired from a security issue on early UNIX systems, that the attacker is not interested in the information “pwd is not read accessible“ which is likely to be the case most of the time during execution. This case of attack also suggests that the attacker wants to know whether “the current run executed by M satisfies ϕ” rather than “a run satisfying ϕ has been executed by M”. In the example, when the login is successful, the attacker knows that `pwd` has been, at some time, read accessible but this is not useful information.

In this thesis, we will investigate several problems concerning opacity properties. In the next section, we present the content of the thesis with an informal explanation of the different problems that have been investigated. We also give some of the intuitions behind the proposed solutions.

1.1 Summary of the Thesis

In the chapter 2, we briefly introduce the models, notations and some general mathematical results that are useful for the subsequent chapters. We emphasis on the notion of complete lattice as we often need to prove the existence of fixpoints using the Knaster-Tarski theorem. We also present the model of Labeled Transition System (LTS), as we consider systems whose behavior can be modeled by LTSs, and present some classical operations on LTSs like parallel composition, complementation, etc.

In chapter 3, we present the notion of opacity introduced in [BKMR08]. We show how this notion of information flow can be deduced from the assumptions that are made about the attacker. More precisely, we consider that the attacker is partially observing the system M via an observation map associating each run of the system to an observation. These attackers are supposedly trying to infer secret information on the basis of the observed traces. As in [BKMR08], we consider that the secret is given as a predicate over the set of possible runs of M, called *secret predicate*. More precisely, the secret information is the occurrence of a run satisfying this secret predicate. Therefore, the attacker should not infer from an observation that the current run of M satisfies the secret predicate. We also consider that the attacker knows the semantics of M and then the set of runs that can explain an observation, i.e. such that the occurrence of one of those runs will imply this observation. In this context, we define the opacity of M with respect to the system and the observation map as the existence, for every observation, of a run explaining this observation and not satisfying the secret predicate. Thus, it is impossible for an inquisitive attacker to infer the truth of the secret predicate as this attacker will always be confused by the possibility that a run not satisfying the secret predicate can have been executed by M.

Then, we present a set of basic results concerning the opacity property. Especially, we consider two kinds of secret predicates: the ones that are true on runs reaching a certain set of states, the state-based predicates, and the ones that are true on runs whose generated trace belongs to a given language, the trace-based predicates. We present how to transform an opacity problem for a kind of predicate to the other. Thus, in the sequel, we can consider the kind of predicate that is the most suitable to solve a particular problem.

In Chapter 4, we investigate the problem of verifying opacity in the context where a subset of the events are observable by the attacker. The chapter 4 consists in three part.

First, we give an algorithm to solve the opacity verification problem that is based on the determinization of M. This determinization procedure directly follows from the definition of opacity. We present this algorithm in a general setting without considering its effective computability, but we show that it allows to compute exactly the set of observed traces such that some secret information is disclosed. For finite systems, the determinization is always possible and then the opacity verification problem is decidable. Going further, we show that the opacity verification problem is PSPACE-complete. We prove this by establishing an equivalence between this problem and the language universality problem for non-deterministic finite automaton.

Second, the procedure to verify opacity, based on the determinization of M, relies on computing the fixpoint of the operator $post_M$, defined from the transitions of M, and is therefore not always effective for infinite state systems with an infinite set of events. For the cases where this determinization is not possible, we consider an Abstract Interpretation framework based on Galois connections to compute an overapproximation of the determinization of M. Then, we show that it allows to infer secret information at runtime on the basis of real observations of M. But we also show that, unlike the case of safety properties, we cannot statically prove the opacity of M by considering this overapproximation framework. Indeed, it may happen that some run satisfying the secret predicate are confused, by the partial observation, by a run, not satisfying the secret predicate, that is generated by the overapproximation but which does not belong to the original semantics of M. Thus, real cases of information flow may not be detected on the overapproximation. On the other hand, an attacker cannot statically compute real attack scenarios from this overapproximation as they cannot be distinguished from false alarms, i.e. observation that are disclosing secret information regarding the overapproximation and that are in fact not possible according to the original behavior of M. More precisely, when an observation possibly discloses secret information according to the overapproximation, the attacker needs to know whether this observation can actually be generated by M to conclude that secret information can be disclosed. However, even if proving the opacity is not possible using overapproximations, we show how to statically prove in some cases the non-opacity of M by computing observations disclosing the secret. This is achieved by solving a reachability problem considering together an overapproximation and an underapproximation of M. The underapproximation, also based on a Galois connection framework, is useful to prove that some observations can effectively be generated by M. This provides a method to automatically search for vulnerabilities and can be interesting for practical applications.

Third, we consider an alternative approach to certify the absence of information flow in the case of possibly infinite state systems but with a finite alphabet of events. The language of M is then not necessarily regular. In that case, we show how to define a monitor for the attacker when a regular abstraction G of the language of M is provided. As for overapproximations based on Galois connections, the attacker cannot statically distinguish false alarm from real attack scenario, but can disclose secret information at runtime on the basis of real observations of M. In this context, as we cannot prove that no information flow will occur in the future, we consider a monitor for the administrator, computed from G, such that all occurrences of information flow will eventually be detected, at last after the occurrence of N events after the attacker discloses the secret, where the bound N is known a priori. For this, we extend the notion of diagnosability for discrete event systems by considering the case when a regular abstraction of a possibly non-regular system is provided. Our main result consists in giving sufficient conditions on the abstraction G such that all occurrences of information flow (on M) will be detected after a bounded delay.

In chapter 5, we investigate the problem of enforcing opacity on a given finite system M. Considering, as in the previous chapter, that the attacker observes a subset of the events of M, we want to enforce the opacity of a finite set of trace-based predicates by supervisory control. The objective of supervisory control is to compute a controller C that is placed in parallel with M such that the composition $C \parallel M$ satisfies a given property [RW87, RW89]. Such a controller only observes a subset of the events of M and can only prevent a subset of these observable events to occur in M. We also search for a controller that is the least restrictive possible. This imposes some constraints over the search space for controllers enforcing the target property. A controlled language is a non-empty and prefix-closed sublanguage of M satisfying the *normality* and *controllability* constraints, i.e. which can be obtained by the parallel composition $C \parallel M$ for a controller C. Classical results from the Ramadge and Wonham theory state that searching for an optimal controller enforcing a property can be reduced to the search for a controlled language satisfying this property that is supremal with respect to the inclusion order relation. We follow an original approach to present the Ramadge & Wonham supervisory control theory, with the objective to treat the opacity control problem as a direct application. The Ramadge & Wonham theory is often presented to implicitly treat trace properties and we extend the presentation to properties on languages. For example, safety is a trace property but opacity and diagnosis are typically not properties on single executions but are expressed in terms of opaque or diagnosable languages.

We also present in a general setting a fixpoint iteration algorithm to compute, when it exists, the supremal controlled language enforcing a given property, e.g safety, opacity or liveness. This fixpoint computation is a classical technique to solve supervisory control problems and consists in applying first the operator giving the supremal sublanguage satisfying the target property. But, as the outcome of this operator is not necessarily a controlled sublanguage of M, i.e. which can be obtained by parallel composition with partial observability and controllability constraints, we apply another operator to obtain its supremal controlled sublanguage. As the two operators are monotone, their composition is also monotone and, according to the Knaster-Tarski theorem, admits then a fixpoint within the complete lattice of prefix-closed sublanguages of $\mathcal{L}(M)$. Starting from $\mathcal{L}(M)$, when a fixpoint is reached, it corresponds to the supremal controlled sublanguage of $\mathcal{L}(M)$ enforcing the property. In the literature, this method has been successfully applied to solve supervisory control problems for example for safety, liveness or non-blocking properties.

At first, we search for sufficient conditions such that the fixpoint computation described above provides a solution to the opacity control problem. These conditions rely on the relationships between three sets of events: the ones observable by the controller, the ones that are controllable and the ones that are observable by the attacker. We show that the fixpoint iteration terminates when the controller observes less events than the attacker or when all the events the attacker can observe can be disabled by control. But, we give an example where the operator iteration technique does not reach a fixpoint after a finite number of iterations. The rest of the chapter is then dedicated to present an alternative approach to solve the opacity control problem. For this, we assume that the controller observes more events than the attacker. In that case, we show that an algorithm to compute the supremal controller when the controller observes all the events also induces an algorithm for the more general situation where the controller is partially observing the events, but still observes more events than the attacker. We therefore assume in the sequel that the control is under total observation. Then, we show that some inherent aspects of the opacity property imply that an optimal control preformed on M only depends on the state of M and on the state estimate of the attacker, i.e. the set of states that are possibly reached with respect to an observed trace. The main consequence of this property is the regularity of the supremal controlled language enforcing the opacity on M, since it admits then only finitely many residual languages. Based on this observation, we propose a "state-based" approach to solve the opacity control problem where the "states" are configurations consisting in the states of the system and the state estimates of the attacker. We finally illustrate this algorithm with an example.

The case where the set of events observable by the attacker and the set of events observable by the controller are not comparable is let for subsequent developments. In fact, the proposed algorithms heavily depend on the fact that the controller observes more than the attacker, or reciprocally, and thus cannot be easily extended to the general case.

The chapter 6 is also devoted to the opacity enforcement problem. But instead of restricting the system into a subset of secure runs, the approach we follow in this chapter consists in modifying the observability of the events at runtime, in order to confuse the attacker and prevent a finite set of secret predicates to be disclosed. This work is an application of the dynamic observer techniques developed in [CT08], with the objective to minimize the set of observable events (sensor activations) such that a diagnosability property is preserved.

In this chapter, we formulate the notion of dynamic projection which is an observation map from the set of words generated by M to a set of observed traces. We consider in this chapter that the observability depends on the traces observed by the attacker. An observability choice maps every trace observed by the attacker to the set of events that are observable after this trace. Then, given an observability choice, we define inductively a dynamic projection which maps a word generated by M to the sequence of events of this word that are observable regarding the observability choice.

In order to preserve the service provided by the system, one might search for a dynamic projection hiding as few events as possible. Unfortunately, we show that the set of dynamic projections enforcing the opacity on M is not closed under union. Therefore, there might exist several optimal solutions that are not comparable. The service that the system is expected to provide is generally expressed in terms of a combination a several kinds of properties. For example, on an e-banking service, one cannot access the account of someone else, but in the same time, one should access its own account if sufficient information concerning the identity is provided. Then, we can search for a dynamic projection enforcing opacity that is supremal with respect to an availability property, for example minimizing the average number of events between a request and an answer. Toward such subsequent developments, we provide in this chapter an algorithm to compute a representation as a finite LTS of the set of all valid dynamic projections, i.e. enforcing opacity. For this, we reduce the problem of computing the set of valid dynamic projections to the problem of computing the set of winning strategies in a safety 2-player game. This game is based on a finite game LTS that is computed from M. As for the control, this game LTS is obtained by showing that the choice of observability that is made to preserve opacity only depends on the state of M and on the state estimate of the attacker. The definition of the game is based on this idea.

In chapter 7, we conclude this thesis by giving a summary of the important aspects and results of this thesis. We also give some possible directions for future works and some perspectives for the techniques developed in this document.

1.2 Related Works

There have been numerous works about the certification of information flow sensitive systems and we cite only some of them here. Some notable early works have been done with the BLP security model in [BL73] which is a formalization of confidentiality policies that were applied in the US military sector. In this article, the authors present a mathematical model of a computer system with confidentiality constraints. This model consists in a set of objects and a set of subjects performing actions on this objects, e.g. read, write, create, delete, etc. The objects and subjects are classified into confidentiality levels ordered from the least to the most confidential, for example:

$$\texttt{public} < \texttt{unclassified} < \texttt{secret} < \texttt{top secret}$$

The access control policy searches to enforce the confidentiality of the data by restricting the possible actions. For example, it is not possible for a subject to read an object that is at a higher level, or a subject cannot write to an object at a lower level. The objective of this policy is to prevent data flow from higher levels, e.g. `top secret`, to lower levels, e.g. `public`. But this model is not sufficient for formal certification as it does not allow to prove the absence of information flow. In fact, a counterexample that appeared on the early multi-user operating systems can be described as follows: a user from the `public` level tries to write to a file `F` at a `top secret` level. If this file `F` does not exist, then the user receives an error message and receives nothing when this file exists. Therefore, in collaboration with another user at the `top secret`, it is possible to create a flow of information from `top secret` to `public` and then bypassing the access control protections. This aspect led some authors to give a more precise definition of information flow properties. Among them, the authors of [GM82] introduced the notion of non-interference which can be stated as follows: considering two security levels, *high* and *low*, and two agents, (users or programs), one at each level, a system is non-interfering if what the high level agent does has no effect on what the low level agent can see.

Several other information flow properties have been proposed after in the literature, e.g. non-inference, separability, restrictiveness, etc, with the objective to model different situations depending on the assumption that are made regarding the secret information and the abilities of the attacker. In [Mcl94], the author proposes a framework to unify these different notions of information flow, called *possibilistic security properties* in the article. This framework is based on classifying these notions with respect to their properties regarding the composition of systems. In [Aba98], the author investigate information flow properties in the context of programming languages. This language-based approach to information is a domain that has been widely investigated since then, with some influent articles like [SM03]. The authors of [FG93] formalize this concept of non-interference using the CSP process algebra. This work is later extended in [RS99] and in [FG00], the authors present a taxonomy of the different notions of information flow using CSP.

In [Maz04], the author proposes to use the notion of opacity to unify different notions of information flow in the context of cryptographic protocols modeled by rewriting rules.

In [BKMR05] and later in [BKMR08], the authors adapt this notion of opacity to transition systems, which eases then the formal comparison between opacity and other notions of information flow. Especially, they show that the problem of verifying some notions of information flow like anonymity (strong or weak as defined in [SS96]) and strong non-deterministic non-interference [FG00] can be formalized as particular instances of opacity for suitable secrets predicates and observation maps. We will not investigate in the present document how opacity can encode these other information properties. We refer to [BKMR08] for a clean and complete comparison between opacity, anonymity, (trace-based) non-interference and non-deducibility. Also, in [Dub09], we show that the opacity property corresponds to the notion of knowledge following the possible worlds model of epistemic logic [Hin62], where the underlying Kripke structure is the set of runs and the equivalence relation is deduced from the observation map. This suggests then that opacity should be expressive enough to formalize a significant set of information flow properties. This work about epistemic logic and opacity is not presented in this document.

In the thesis, we follow two objectives: verifying the opacity property on a system, or at least detect some counterexamples, and enforcing opacity on an insecure system. There have been several publications investigating opacity verification problems. In [SH08], the authors present a verification procedure for initial opacity which is a special case of opacity where the secret predicates depend on the first state of the runs. In [AČZ06], the authors investigate the verification problem for secrecy, which is actually the conjoint opacity of a secret predicate and its negation. They show that secrecy cannot be expressed as a μ-calculus formula over the traces of the system. In [AČC07], this problem is circumvented by defining temporal logics that are interpreted along the paths by agglomerating states that are observationally equivalent. In chapter 4, we present a verification procedure that is based on computing the determinization of M. The work presented in [AČC07] seems to have similarities with temporal logics that are interpreted over the Kripke structure $M \parallel det(M)$, where the state estimates of the attacker are given by the states of $det(M)$. A formal comparison between the two approaches is let for subsequent developments. Finally, we show that in the case of finite systems, the opacity verification problem is equivalent to the language universality problem. Therefore, the techniques developed in [DDMR08b, DDMR08a] should provide more efficient verification algorithms that the determinization procedure presented in Chapter 4. We use an abstract interpretation framework to investigate the opacity verification problem. This has already been investigated in [Mas05] for non-interference properties. In this thesis, the author investigates sufficient conditions about the system and the abstractions (given as closure operators) such that the non-interference is satisfied. We follow another approach in Chapter 4. We chose the point of view of an attacker and study how an abstract interpretation framework can be applied to infer secret information. Our motivation for this approach is to consider a system as secure as long as no security breach has yet been discovered. Indeed, the search for sufficient condition for opacity may reject systems such that no computable attack scenarios can be computed in practice. Our approach is therefore complementary to the one of [Mas05].

Applying supervisory control to enforce confidentiality properties is an emerging field of research. In [Ric06], the author adapts the decentralized supervisory control theory in order to ensure the Chinese Wall Policy. In [CMR07a, CMR07b], the authors investigate the control problem for non-interference properties. This work has later been extended to timed systems in [BCLR09]. The opacity control problem has first been introduced in [BBB$^+$06] and later developed in [BBB$^+$07]. In these articles, the authors consider the case of several attackers, each of them trying to infer the truth of a secret predicate (one predicate per attacker). They investigate the control problem with the hypothesis that the controller observes all events and all events can be controlled. They provided sufficient conditions for the existence and effective computability of a supremal controller enforcing the opacity of all the secrets together. The objective of the chapter 5 is to extend this work by releasing the assumptions that all the events are controllable and observable by the controller. We consider in this chapter the case of one controller and a finite set of secret predicates. Another work investigating the opacity control problem has been published in [TO08]. In this article, the authors consider that all the events are observable but are not all controllable. With some assumptions on the set of observable events of the attacker and the controllable events, they show that the fixpoint computation of Ramadge and Wonham provides the optimal solution to the opacity control problem. We will more precisely compare this work with our approach in Chapter 5.

The work of chapters 5 and 6 also has some relationships with the earlier work done by Schneider on security automata [Sch00], subsequently extended to edit automata in [LBW05]. The goal pursued in [Sch00] is to produce an interface automata that enforces a security policy, consisting of integrity properties, represented by a prefix-closed language of safe executions: the interface automaton rejects the inputs from the environment that would lead the system to leave the subset of safe execution. In [LBW05], the author consider several kinds of automata, called edit automata, classified with respected to their transformational capabilities, e.g. halt system, remove actions, insert actions, etc. Then, they give a set-theoretic characterization of the security policies that can be enforced by each category of edit automata. These enforcement monitoring techniques have also been investigated in [FFM09]. In this paper, the authors give a classification of safety properties that can be enforced by monitoring. They also provide an algorithm to compute an enforcing monitor given an automaton accepting the safety property.

There have also been some related works in the context of model-based testing for security properties. In this context, some techniques have been developed to test access control policy with partial specification [DFG$^+$06]. In [MDJ09], the work presented in Chapter 5 is applied to test the implementation of integrity and confidentiality policies. In this paper, the test generation and selection is led by an ideal access control that is computed from the specification.

Finally, as systems with confidentiality requirements often use randomization in order to decrease the likelihood of information leakage, there have been some works considering information flow properties in the context of probabilistic systems. In this direction, the authors of [LM05] extend the notion of opacity to the case of probabilistic system. Also, in [CPP08, BCP08], the authors consider security protocols as noisy channels and therefore use techniques from information theory to analyze the quantity of secret information that flow from the system to an attacker. A similar approach is considered in [Low02] using CSP.

1.3 Contributions

In this section, we summarize the contributions of this thesis for each chapter.

In the first part of the chapter 4, in Section 4.3, we present how an abstract interpretation framework based on Galois connection can be used to solve the opacity problem. We show how an attacker can infer secret information on a possibly infinite system by abstracting the states and the events of the system. We also show how to automatically detect vulnerabilities by considering together underapproximations and overapproximations. The possible applications of these techniques can be to automate the detection of security flaws within programs. These results have been published in [Dub09]. In the second part of this chapter, in Section 4.4, we present a new approach to certify the absence of information leakage. In this section, we consider a regular abstraction G of the system and combine opacity with diagnosis theory. We first show that it cannot be decided from an regular abstraction if M is opaque or not, but we show that an attacker may infer secret information from this abstraction on the basis of real observations of M. The purpose is to decide whether the occurrences of information flow will eventually be detected by a monitor, from an administrator point of view, also partially observing the system. For this, we extend the diagnosability theory in the context of possibly infinite systems when a regular abstraction is provided. This work extend the results of the publications of [DJM07] and [DJM09]. The theory presented in the chapter 4 has also been the occasion of the internship of Wassim Wehbi (ESIB, Lebanon) co-supervised with Hervé Marchand. The project consisted in the development of a prototype of a tool to compute the sound monitor of Chapter 4 and to verify the diagnosability of a safety property on an LTS. We have searched in this internship to use the module system of OCaml to create a tool that can be easily extended to different datatypes and different algorithms, e.g. backward analysis or forward analysis for reachability.

The opacity control problem, treated in chapter 5, has not been much investigated in the literature. Compared to [BBB$^+$07] and [TO08], we consider the problem with more general hypothesis regarding the observability of the events for the controller and the attacker. In that case, we show that opacity is out of the scope of the classical Ramadge and Wonham fixpoint iteration techniques. So we develop a new algorithmic approach to solve the opacity control problem. We provide a solution when the alphabet of events observable by the attacker and the ones observable by the controller are comparable. The general case is let for subsequent developments. The complexity of the opacity control problem is also not investigated here. Note also that we slightly generalize the Ramadge and Wonham theory with the objective to enforce properties over languages, like opacity or diagnosis, whereas it is usually presented to enforce trace properties. These results have been published in [DDM08, DDM09]

In chapter 6, we follow a new approach to enforce opacity properties. The notion of dynamic projections has been introduced in [CT08] for diagnosability enforcement. But as the opacity and diagnosis are not comparable notions, new algorithms have been developed for opacity. In [CT08], the solution to diagnosis enforcement is based on a reduction to a safety 2-player game. We follow the same technique here but with a new game definition. We also formalize more precisely than in [CT08], the notion of observer and dynamic projections, and how they can be related to each other via the notion of observability choice. This work have been published in [CDM09a]. We follow here a slightly different presentation as we consider different assumptions about the observation abilities of the attacker.

2 Basic Notions

In this chapter, we will provide the notations and some basic results that will be used throughout the thesis. We will present the labeled transition system model and some basic associated model transformations. In the subsequent chapters, we will often formalize the problems and provides solution expressed in terms of fixpoint computation. We then start this section with the useful concepts of functions and operators within partially ordered sets and lattices.

2.1 Sets and Relations

In this section, we present some notions and notations about sets and relations. This presentation and the notations that we use troughout the thesis are inspired from [BS81]. Given two sets A and B, one denote by $A \times B = \{(x, y) : x \in A, y \in B\}$ the Cartesian product of A and B. Given a set A, the powerset of A is denoted $\mathcal{P}(A) = \{X \subseteq A\}$.

Definition 2.1 (Function) *A function f from A to B, written $f : A \to B$ is a subset of $A \times B$ such that for each $x \in A$, there exists at most one $y \in B$ with $(x, y) \in f$. We denote such y by $f(x)$. The set A is the* domain *of f and B its* codomain*.*

We use the term *partial function* to emphasis that $f(x)$ may be undefined for some $x \in A$. We say that f is a *total function*, also called a *map*, when $f(x)$ exists for every $x \in A$. Given a subset $X \subseteq A$, $f(X) = \{f(x) : x \in X\}$. We say that f is *injective* if for all $x, x' \in A$, $f(x) = f(x') \implies x = x'$ and *surjective* if for all $y \in B$, there exists $x \in A$ such that $f(x) = y$. The *range* of f is the set $f(A) \subseteq B$ and f is surjective if and only if $B = f(A)$. Also, f is *bijective* when f is injective and surjective. For $Y \subseteq B$, $f^{-1}(Y) = \{x \in A : f(x) \in Y\}$ which also defines a map $f^{-1} : \mathcal{P}(B) \to \mathcal{P}(A)$. For $f : A \to B$ and $g : B \to C$, the *composition* of g and f is the function $g \circ f : A \to C$ defined by $g \circ f(x) = g(f(x))$.

In the sequel, we will often need to study iterations of functions. We denote by $id_A : A \to A$ the identity function defined for every $x \in A$ by $id_A(x) = x$. Let $f : A \to A$ be a map. The iterations of f are defined by $f^0 = id_A$ and for $n \in \mathbb{N}$, $f^n = f^{n-1} \circ f$. An element $x \in A$ is a *fixpoint* of f if $f(x) = x$. Finally, we say that a map f is idempotent if, $f \circ f = f$.

We will now give some definitions about relations. We briefly introduce equivalence relations. In the thesis, we will regularly use results and constructions coming from lattice theory, especially the theorem of Knaster-Tarski stating the existence and characterization of least/greatest fixpoints. We then introduce in this section order relations, posets and complete lattices.

Definition 2.2 (Relation) *Let A be a set. A* binary relation *r over A is a subset of $A \times A$.*

We say that this relation is

- *reflexive* if $\forall x \in A, (x, x) \in r$
- *symmetric* if $\forall x, y \in A, (x, y) \in r \Rightarrow (y, x) \in r$
- *antisymmetric* if $\forall x, y \in A, (x, y) \in r \wedge (y, x) \in r \Rightarrow x = y$
- *transitive* if $\forall x, y, z \in A, (x, y) \in r \wedge (y, z) \in r \Rightarrow (x, z) \in r$

Often in the thesis, we will use the notation $r(x)$ for the set of elements of A that are related to x. In other words, a relation r over A also defines a map $r : A \to \mathcal{P}(A)$.

Definition 2.3 (Equivalence relation) *A relation θ over A is called an* equivalence relation *if θ is reflexive, symmetric and transitive.*

We use the notation $x \sim_\theta y$ for $(x, y) \in \theta$ and $\theta(x) = \{y \in s : y \sim_\theta x\}$ for the *equivalence class* of x. We will also use the notation $[x]_\theta = \theta(x)$ for the equivalence class. We denote by $A/_\theta = \theta(A) = \{\theta(x) : x \in A\}$ the *quotient set* of A by θ which is the set of equivalence classes of A. We denote by $Eq(A)$ the set of equivalence relations over A.

Theorem 2.1 (Quotient map) *Given a surjective map $f : A \to B$ and an equivalence relation θ such that $(x, y) \in \theta$ implies $f(x) = f(y)$, then there exists a unique map $f_\theta : A/_\theta \to B$ such that $f = f_\theta \circ \theta$. The map f_θ is called the quotient map of f by θ.*

Definition 2.4 (Equivalence kernel) *Given a map $f : A \to B$, the* equivalence kernel *of f is the equivalence relation θ_f defined by $(x, y) \in \theta_f$ if $f(x) = f(y)$.*

Proposition 2.1 (Canonical quotient) *Given a map $f : A \to B$, the* canonical quotient map *is the map $can(f) : A/_{\theta_f} \to f(A)$, obtained by applying theorem 2.1 with the equivalence kernel θ_f. With this construction, the map $can(f)$ is bijective.*

2.1.1 Posets

Definition 2.5 (Order relation) *A relation $\sqsubseteq$ over a set A is called an* order relation *if $\sqsubseteq$ is reflexive, antisymmetric and transitive.*

Classical examples of partial orders are $\leq$ over $\mathbb{N}$, or $\subseteq$ over $\mathcal{P}(A)$ for any set A. We note $x \sqsubset y$ when $x \sqsubseteq y$ and $x \neq y$.

Definition 2.6 (Poset) *If $\sqsubseteq$ is an order relation over A, then $(A, \sqsubseteq)$ is called a* partially ordered set*, also often called a* poset.

Given a map $f : A \to A$ where $(A, \sqsubseteq)$ is a poset, we say that f is *extensive* if for all $x \in A$, $x \sqsubseteq f(x)$ and that f is *reductive* if for all $x \in A$, $f(x) \sqsubseteq x$.

Let $(A, \sqsubseteq_a)$ and $(B, \sqsubseteq_b)$ be two posets and $f : A \to B$. The function f is *monotone* if for all $x, y \in A$, $f(x) \sqsubseteq_b f(y)$ whenever $x \sqsubseteq_a y$.

Definition 2.7 (Closure operator) *Let $(A, \sqsubseteq)$ be a poset. Given a map $\rho : L \to L$, ρ is an* upper closure operator *over L if ρ is monotone, idempotent and extensive. Also, ρ is a* lower closure operator *over L if ρ is monotone, idempotent and reductive.*

Example 2.1 *Given a set A and an equivalence relation $\theta \in Eq(A)$, the map $f : \mathcal{P}(A) \to \mathcal{P}(A)$, $X \mapsto \cup\theta(X)$ is an upper closure operator.*

Definition 2.8 (Galois connections) *Given two posets $(A, \sqsubseteq_a)$ and $(B, \sqsubseteq_b)$, two functions $\alpha : A \to B$ and $\gamma : B \to A$ establish a Galois connection between A and B, denoted $(A, \sqsubseteq_a) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (B, \sqsubseteq_b)$, when for all $x \in A$ and $y \in B$,*

$$\alpha(x) \sqsubseteq_b y \iff x \sqsubseteq_a \gamma(y)$$

In that case, both α and γ are monotone, the map $\gamma \circ \alpha : A \to A$ is extensive and $\alpha \circ \gamma : B \to B$ is reductive.

A sequence $\{x_i\}_{i \in \mathbb{N}}$ of elements of A is an increasing chain if for all $i \in \mathbb{N}$, $x_i \sqsubseteq x_{i+1}$. We say that the chain is strictly increasing if for all $i \in \mathbb{N}$, $x_i \sqsubset x_{i+1}$. For $X \subseteq A$ and $y \in A$ we say that y is an upper bound (resp. lower bound) if for all $x \in X$, $x \sqsubseteq y$ (resp. $y \sqsubseteq x$). Also, y is a least upper bound, if for every other upper bound z, we have $y \sqsubseteq z$. When such a least upper bound exists, it is unique and denoted $\sqcup X$. Similarly, $\sqcap X$ denotes the greatest lower bound.

2.1.2 Lattices

We introduce now the notion of lattice and some important results associated to lattices.

Definition 2.9 *A partially ordered set $(L, \sqsubseteq)$ is called a* complete lattice *whenever $\sqcup X$ and $\sqcap X$ exist for every $X \subseteq L$.*

We directly define the notion of complete lattice as it is the only one that will be considered in the document. In the following, the term *lattice* will implicitely mean *complete lattice.*

A lattice is often denoted by $(L, \sqsubseteq, \sqcap, \sqcup, \bot, \top)$, where $\top = \sqcup L$ and $\bot = \sqcap L$ denote respectively the greatest and the least element of L.

Example 2.2 *If A is a set, then $(\mathcal{P}(A), \subseteq, \cap, \cup, \emptyset, A)$, the set of subsets of A together with the set inclusion for order relation, forms a complete lattice.*

Example 2.3 *If A is a set, then $(Eq(A), \subseteq, \cap, \vee, \{(x,x) : x \in A\}, A \times A)$ forms a complete lattice where for $\Theta \subseteq Eq(A)$, $\vee\Theta$ is defined by: for all $(x,y) \in A \times A$ $(x,y) \in \vee\Theta$ when there exists a finite sequence $x_0, x_1, \ldots x_n \in A$ such that $x_0 = x$, $x_n = y$ and for every $i, 0 \leq i < n$, there exists $\theta \in \Theta$ such that $(x_i, x_{i+1}) \in \theta$.*

Example 2.4 *The set of intervals of $\mathbb{R}$, $(I(\mathbb{R}), \subseteq, \cap, \sqcup, \emptyset, \mathbb{R})$, where for $X \subseteq I(\mathbb{R})$, $\sqcup X = [min(\{a : \exists b, [a,b] \in X\}), max(\{b : \exists a, [a,b] \in X\})]$, is a complete lattice.*

The height of a lattice L, denoted $height(L)$, is the maximum number of pairwise disjoint elements in a chain on L. Typically, $height(L) < \infty$ when $|L| < \infty$.

Several times in this thesis, we will need to compute the upper bound of the set of iterations of a function, i.e. the element $\sqcup\{f^i(x) : i \in \mathbb{N}\}$. We formally define the needed operation as follows.

Definition 2.10 *Let $(L, \sqsubseteq, \sqcap, \sqcup, \bot, \top)$ be a complete lattice. Given a map $f : L \to L$, we define $f^\uparrow : L \to L$ by $f^\uparrow(x) = \sqcup\{f^i(x) : i \in \mathbb{N}\}$. Similarly, $f^\downarrow(x) = \sqcap\{f^i(x) : i \in \mathbb{N}\}$.*

Note that $f^\uparrow$ is monotone whenever f is monotone. We will see now some conditions under which $f^\uparrow$ can be effectively computed.

First, we present the notion of ω-continuity (or Scott continuity). Considering two complete lattices $(L_1, \sqsubseteq_1, \sqcap_1, \sqcup_1, \bot_1, \top_1)$, $(L_2, \sqsubseteq_2, \sqcap_2, \sqcup_2, \bot_2, \top_2)$ and a map $f : L_1 \to L_2$. We say that f is *ω-continuous* if for every increasing chain $\{x_i\}_{i \in \mathbb{N}}$ over L_1, $f(\sqcup_1\{x_i : i \in \mathbb{N}\}) = \sqcup_2\{f(x_i) : i \in \mathbb{N}\}$.

Second, we introduce the notion of least and greatest fixpoints. Given a complete lattice $(L, \sqsubseteq, \sqcap, \sqcup, \bot, \top)$ and a map $f : L \to L$. An element x is a least fixpoint of f if x is a fixpoint and is smaller that any other fixpoint of f. By antisymmetry, such fixpoint is unique and denoted $lfp(f)$. The greatest fixpoint is similarly defined and denoted $gfp(f)$. The following theorem states some condition for a given map to admit a least or a greatest fixpoint.

Theorem 2.2 (Knaster-Tarski [Tar55]) *Let $(L, \sqsubseteq, \sqcap, \sqcup, \bot, \top)$ be a complete lattice and $f : L \to L$.*

- *If f is monotone, then there exists a least and a greatest fixpoint of f.*
- *$lfp(f) = \sqcup\{x \in L : f(x) \sqsubseteq x\}$ and $gfp(f) = \sqcap\{x \in L : x \sqsubseteq f(x)\}$.*
- *If f is also ω-continuous, then $lfp(f) = \sqcup\{f^i(\bot) : i \in \mathbb{N}\}$ and $gfp(f) = \sqcap\{f^i(\top) : i \in \mathbb{N}\}$.*

If $height(A) < \infty$, then there exists $N \in \mathbb{N}$ such that $lfp(f) = f^N(\bot)$. Similarly, the *greatest fixed point* $gfp(f) = f^N(\top)$ for some $N \in \mathbb{N}$.

The previous theorem provides a way to compute $f^\uparrow$ that is detailed in the next proposition.

Proposition 2.2 *Let $(L, \sqsubseteq, \sqcap, \sqcup, \bot, \top)$ be a complete lattice and $f : L \to L$ be monotone and ω-continuous. Then, for all $x \in L$, $f^\uparrow(x) = lfp(y \mapsto x \sqcup f(y))$.*

Proof. Let $x \in L$ and define $y = \sqcup\{f^n(x) : n \in \mathbb{N}\}$ and $g_x : L \to L$, $z \mapsto x \sqcup f(z)$.

- $y \sqsubseteq gfp(g)$.

  Indeed, define the increasing chain $\{y_n\}_{n\in\mathbb{N}}$ by $y_n = \{f^k(x) : k \leq n\}$. Since, $x \sqsubseteq x \sqcup f(\bot)$, $y_0 \sqsubseteq g(\bot)$. Also, if we assume that $y_n \sqsubseteq g^{n+1}(\bot)$, then $f(y_n) \sqsubseteq f(g^{n+1}(\bot))$ because f is monotone. So, $x \sqcup f(y_n) \sqsubseteq x \sqcup f(g^{n+1}(\bot))$. But, $\{f^{k+1}(x) : k \leq n\} \sqsubseteq f(y_n)$, so $y_{n+1} \sqsubseteq x \sqcup f(y_n)$. Then, $y_{n+1} \sqsubseteq g^{n+2}(\bot)$. Since f is monotone and ω-continuous, then so is g_x. Then, for all $n \in \mathbb{N}$, $y_n \sqsubseteq lfp(g)$. Finally, $y \sqsubseteq lfp(g)$.

- y is a fixpoint of g.

$$\begin{aligned} g(y) &= g(\sqcup\{f^n(x) : n \in \mathbb{N}\}) \\ &= x \sqcup f(\sqcup\{f^n(x) : n \in \mathbb{N}\}) \\ &= x \sqcup (\sqcup\{f^{n+1}(x) : n \in \mathbb{N}\})(\text{since f is } \omega\text{-continuous}) \\ &= y \end{aligned}$$

So we conclude that $y = lfp(g)$. □

Finally, we define the notion of sublattice of a lattice.

Definition 2.11 (Sublattice) *Let $(L, \sqsubseteq, \sqcap, \sqcup, \bot, \top)$ be a lattice. A sublattice of L is a subset $H \subseteq L$ such that the operations $\sqcap$ and $\sqcup$ induce a lattice structure on H: for all $X \subseteq H$, $\sqcap X \in H$ and $\sqcup X \in H$.*

2.2 Labeled Transition Systems

In this section, we introduce the notation that will be used to represent a labeled transition system M.

Let Λ be a possibly infinite set of labels representing the events of a system. The notations $\lambda, \lambda', \dots$ denote typical elements of Λ, called the alphabet of M. In this document, we will use the notation Σ to emphasis when this alphabet of events is finite. The symbols $\sigma, \sigma', \dots$ will then denote typical elements of Σ.

If A and B are two subsets of Λ, then $AB = \{\lambda\lambda' : \lambda \in A, \lambda' \in B\}$. We denote $\Lambda^n = \Lambda^{n-1}\Lambda$, assuming that $\Lambda^0 = \{\epsilon\}$, with ϵ being the empty word. Let $\Lambda^* = \cup_{n\in\mathbb{N}}\Lambda^n$ denote the set of finite words over Λ. A set L of finite words over Λ, i.e. $L \subseteq \Lambda^*$, is called a language over Λ. The concatenation of two words $w = \lambda_1\lambda_2\dots\lambda_n$ and $w' = \lambda'_1\lambda'_2\dots\lambda'_m$ is the word $ww' = \lambda_1\lambda_2\dots\lambda_n\lambda'_1\lambda'_2\dots\lambda'_m$. This is extended to languages: given $L, L' \subseteq \Lambda^*$, $LL' = \{ww' : w \in L, w' \in L'\}$.

For $w \in \Lambda^*$, $|w|$ is called the length of w, i.e. the number of letters occurring in w. The length of the empty word is zero and $|ww'| = |w| + |w'|$.

Given two words $w, w' \in \Lambda^*$, we say that w is a *prefix* of w' if there exists $w'' \in \Lambda^*$ such that $w' = ww''$. Prefix is an order relation over Λ^* and is also be denoted $w \leq w'$. Let L be a language over Λ, i.e. $L \subseteq \Lambda^*$. The set of prefixes of L is denoted $pref(L) = \{w \in \Lambda : \exists w' \in L, w \leq w'\}$. We will say that L is *prefix-closed* when $L = pref(L)$ and that L is *extension-closed* when $L = L\Lambda^*$.

Given a language $L \subseteq \Lambda^*$, the residuation is the inverse operation of the concatenation and associates to each word of $w \in \Lambda^*$ the set of words w' extending w in such a way that ww' remains in L. This language is denoted $w^{-1}L$ and is formally defined by the map $\Lambda^* \to \mathcal{P}(\Lambda^*),\ w \mapsto w^{-1}L = \{w' \in \Lambda^* : ww' \in L\}$.

For a subset $\Lambda' \subseteq \Lambda$, we will use the term projection for the map $\pi_{\Lambda\to\Lambda'}$ removing from a word of Λ^* all labels that are not in Λ':

$$\begin{array}{rccl} \pi_{\Lambda\to\Lambda'} : & \Lambda^* & \to & \Lambda'^* \\ & \epsilon & \mapsto & \epsilon \\ & w\lambda & \mapsto & \begin{cases} \pi_{\Lambda\to\Lambda'}(w)\lambda \text{ if } \lambda \in \Lambda' \\ \pi_{\Lambda\to\Lambda'}(w) \text{ otherwise} \end{cases} \end{array}$$

Let S be a possibly infinite set of labels representing the possible states, or configurations, of a system M, with $s, s' \in S$ denoting the elements of S. For example, if a finite set of variables is used within a program P then S can represent all the possible valuations of these variables in the semantics of P. Also, in the reminder of the document, the notation Q will be used to emphasis that the set of states is finite and $q, q', \cdots$ will range for elements of Q.

A finite *run* (also called execution) is a sequence of the form:

$$r = s_0 \overset{\lambda_1}{\rightarrow} s_1 \overset{\lambda_2}{\rightarrow} s_2 \ \cdots \ s_{n-1} \overset{\lambda_n}{\rightarrow} s_n \tag{2.1}$$

constructed by alternating states of S and events of Λ. The set of all finite runs which can be constructed from Λ and S is denoted $E(\Lambda, S) = S(\Lambda S)^*$. The states of S are identified in $E(\Lambda, S)$ to the elements of $\{s\epsilon : s \in S\}$.

Example 2.5 *A run is a discrete representation of the dynamic evolution of a process. For example, we can model the journey to attend a conference in London as the sequence of geographical positions and transportation information. Starting from Rennes we can choose to take the train to go to Paris and then the plane to London:*

$$\mathit{Rennes} \overset{\mathit{take\ the\ train}}{\longrightarrow} \mathit{Paris} \overset{\mathit{take\ the\ plane}}{\longrightarrow} \mathit{London}$$

Another possibility for the traveler can be to go to Paris by car. If the airport prevents any plane to take off because of threatening weather forecasts, the traveller can then take the train to London:

$$\mathit{Rennes} \overset{\mathit{take\ the\ car}}{\longrightarrow} \mathit{Paris} \overset{\mathit{plane\ canceled}}{\longrightarrow} \mathit{Paris} \overset{\mathit{take\ the\ train}}{\longrightarrow} \mathit{London}$$

We remark that the traveler can always choose not to take the train and to go to Paris by car but cannot choose that planes are not canceled. For the traveler, the event "take the car" is controllable whereas "plane canceled' is not. This notion will be formally defined in Chapter 5 where it plays a central rôle in the possibility to enforce the security on a critical system.

The operator $tr : E(\Lambda, S) \rightarrow \Lambda^*$ gives the trace of a run, i.e. the sequence of events occurring in this run. For example, the trace of the run in (2.1) is the word $tr(r) = \lambda_0\lambda_1\lambda_2 \ldots \lambda_n$. The length of a run is defined by $length(r) = |tr(r)|$. For $i \in \mathbb{N}$, we denote by $r(i)$ the state at position i in r. The operators $fst(r) = r(0)$ and $lst(r) = r(length(r))$ denote respectively the first and the last state of a run r. For r in (2.1), $fst(r) = s_0$ and $lst(r) = s_n$.

The concatenation of two runs $r = s_0 \overset{\lambda_1}{\rightarrow} s_1 \overset{\lambda_2}{\rightarrow} s_2 \ \cdots \ s_{n-1} \overset{\lambda_n}{\rightarrow} s_n$ and $r' = s'_0 \overset{\lambda'_1}{\rightarrow} s_1 \overset{\lambda'_2}{\rightarrow} s_2 \ \cdots \ s'_{m-1} \overset{\lambda'_m}{\rightarrow} s'_m$ is defined when $s_n = s'_0$ and is the run $r \cdot r' = s_0 \overset{\lambda_1}{\rightarrow} s_1 \ \cdots \ s_{n-1} \overset{\lambda_n}{\rightarrow} s'_0 \overset{\lambda'_1}{\rightarrow} s_1 \ \cdots \ s'_{m-1} \overset{\lambda'_m}{\rightarrow} s'_m$. We say that r is a prefix of r', denoted $r \leq r'$, when there exists

$r'' \in E(\Lambda, S)$ such that $r' = r \cdot r''$. In that case, we also say that r' is an extension of r. The relation $\leq$ is an order relation over $E(\Lambda, S)$.

The behavior of a system are modeled as a set of runs. To ease the representation of this set, we will use the LTS model defined below.

Definition 2.12 *An LTS is a quadruple $M = (\Lambda, S, \delta, S_0)$ where:*

- *Λ is an alphabet of events;*
- *S is a set of states;*
- *$\delta : \Lambda \times S \to \mathcal{P}(S)$ is the transition function;*
- *$S_0 \subseteq S$ are the initial states.*

An LTS can be seen as a labeled directed graph where the vertices represent the states and the edges the events.

Example 2.6 *The graph depicted in Figure 2.1 represent an LTS where the alphabet of events is $\Lambda = \{a, b\}$ and the set of states is $S = \{s_0, s_1, s_2, s_3\}$. The initial state is the singleton s_0, i.e. every execution starts from s_0. The fact that $s_2 \in \delta(a, s_0)$ means that when the system starts, it can reach s_2 when a occurs. This is pictured by an arc labeled a from s_0 to s_2 in the graph.*

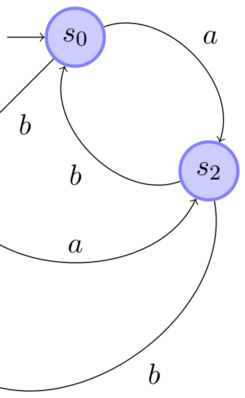


Figure 2.1: An example of LTS

We say that the LTS M is deterministic if $|S_0| = 1$ and for all $\lambda \in \Lambda$ and $s \in S$, $|\delta(\lambda, s)| \leq 1$. For example, the LTS of Figure 2.1 is not deterministic as $\delta(b, s_2) = \{s_0, s_3\}$. When the LTS M is deterministic, the transition function can be given as a partial function $\delta : \Lambda \times S \to S$ which can then be extended to sets of words $\delta : \Lambda^* \times S \to S$ by $\delta(\epsilon, s) = s$ and $\delta(w\lambda, s) = \delta(\lambda, \delta(w, s))$ for $s \in S$, $w \in \Lambda^*$ and $\lambda \in \Lambda$. In the sequel, it will often be implicit that such transition function of a deterministic LTS can be extended to words in this way.

Let X, Y be two set of states of S.

- The set of executions of M starting from X and ending in Y is the set $\mathcal{R}(M, X, Y)$ consisting of the set of runs $r = s_0 \xrightarrow{\lambda_1} s_1 \xrightarrow{\lambda_2} s_2 \cdots s_{n-1} \xrightarrow{\lambda_n} s_n \in E(\Lambda, S)$ such that $s_0 \in X$, $s_n \in Y$ and for all $i, 0 \leq i < n$, $s_{i+1} \in \delta(\lambda_i, s_i)$.
- The set of executions of M starting from states in X is denoted $\mathcal{R}(M, X) = \mathcal{R}(M, X, S)$.
- Finally, we use the notations $\mathcal{R}(M) = \mathcal{R}(M, S_0)$, for the set of runs of M.

Remark 2.1 *Following Theorem 2.2, $\mathcal{R}(M, X)$ can also be defined by $\mathcal{R}(M, X) = lfp(f_X)$, where the function*

$$\begin{array}{rccl} f_X : & \mathcal{P}(E(\Lambda, S)) & \to & \mathcal{P}(E(\Lambda, S)) \\ & R & \mapsto & X \cup \{r \xrightarrow{\lambda} s : r \in R,\ \lambda \in \Lambda,\ s \in \delta(\lambda, lst(r))\} \end{array}$$

is monotone on the complete lattice $\mathcal{P}(E(\Lambda, S))$.

Similarly to the set of runs, we can define the language generated by an LTS.

- Let $\mathcal{L}(M, X, Y) = tr(\mathcal{R}(M, X, Y))$ denote the set of traces of runs from X to Y.
- Let $\mathcal{L}(M, X) = \mathcal{L}(M, X, S)$ denote the language generated by M starting from X.
- Finally, let $\mathcal{L}(M) = \mathcal{L}(M, S_0, S)$ denote the language of M.
- For a language $L \subseteq \Lambda^*$. If $L = \mathcal{L}(M, S_0, Y)$, we say that L is accepted by M and the accepting states Y.

The notion of regular languages will play a central rôle in this thesis, and is defined below.

Definition 2.13 (Automaton) *A (finite) automaton is a tuple $G = (\Sigma, Q, \delta_G, Q_0, Q_f)$ where $(\Sigma, Q, \delta_G, Q_0)$ is a finite LTS, i.e. Σ and Q are finite, with a set of accepting states $Q_f \subseteq Q$.*

In the sequel, the acronym DFA will stand for Deterministic Finite Automaton.

Definition 2.14 (Regular language) *Let Σ be a finite alphabet. A language $L \subseteq \Sigma^*$ is regular if there exists an automaton $G = (\Sigma, Q, \delta_G, Q_0, Q_f)$ such that $\mathcal{L}(G, Q_0, Q_f) = L$.*

We introduce now the notion of completeness which states that some events can always occur in M. The complementation procedure gives a complete LTS from an incomplete one.

Definition 2.15 (Completeness) *Let $\Lambda' \subseteq \Lambda$. We say that M is Λ'-complete when for all $\lambda \in \Lambda'$, $s \in S$, $\delta(\lambda, s) \neq \emptyset$. We say that M is complete when M is Λ-complete.*

Definition 2.16 (Complementation) *Given an LTS $M = (\Lambda, S, \delta, S_0)$, the complementation of M is the LTS $comp(M) = (\Lambda, S \cup \{q\}, \delta_c, S_0)$ where $q \notin S$ and:*

$$\begin{array}{rccl} \delta_c : & \Lambda \times (S \cup \{q\}) & \rightarrow & \mathcal{P}(S \cup \{q\}) \\ & \lambda,\ s & \mapsto & \begin{cases} \delta(\lambda, s) \text{ when not empty} \\ \{q\} \text{ when } \delta(\lambda, s) = \emptyset \text{ or } s = q \end{cases} \end{array}$$

The LTS $comp(M)$ is such that $\mathcal{L}(comp(M)) = \Lambda^*$ but preserves the languages accepted in M, i.e. if $F \subseteq S$, $\mathcal{L}(comp(M), S_0, F) = \mathcal{L}(M, S_0, F)$.

In practice, systems are often composed of several simpler sub-systems. In the context of systems modeled by LTSs, we consider the parallel composition defined below, representing the concurrent behavior of two or more LTS synchronizing on common events.

Definition 2.17 (Parallel composition) *The parallel composition $M^1 \parallel M^2 = (\Lambda, S, \delta, s_0)$ of two LTSs $M^1 = (\Lambda^1, S^1, \delta^1, s_0^1)$ and $M^2 = (\Lambda^2, S^2, \delta^2, s_0^2)$ is defined by:*

- *$\Lambda = \Lambda^1 \cup \Lambda^2$ and $S = S^1 \times S^2$*
- *$s_0 = (s_0^1,\ s_0^2)$*
- *$\delta : \Lambda \times S \rightarrow \mathcal{P}(S)$ is defined by:*

$$\lambda, (s_1, s_2) \quad \mapsto \quad \begin{cases} \delta^1(\lambda, s_1) \times \delta^2(\lambda, s_2) \text{ if } \lambda \in \Lambda^1 \cap \Lambda^2 \\ \{s_1\} \times \delta^2(\lambda, s_2) \text{ if } \lambda \in \Lambda^1 \setminus \Lambda^2 \\ \delta^1(\lambda, s_1) \times \{s_2\} \text{ if } \lambda \in \Lambda^2 \setminus \Lambda^1 \end{cases}$$

Now, let $\pi_1 = \pi_{\Lambda \rightarrow \Lambda^1}$ and $\pi_2 = \pi_{\Lambda \rightarrow \Lambda^2}$ to characterize the languages generated by the parallel composition.

Proposition 2.3 *Let M^1, M^2 and M as in Proposition 2.17, then*

$$\mathcal{L}(M) = \ \pi_1^{-1}(\mathcal{L}(M^1)) \ \cap \ \pi_2^{-1}(\mathcal{L}(M^2))$$

Now, if $F^i \subseteq S^i$, $i = 1, 2$ and $F = F^1 \times F^2$, then,

$$\mathcal{L}(M, S_0, F) = \ \pi_1^{-1}(\mathcal{L}(M^1, S_0^1, F^1)) \ \cap \ \pi_2^{-1}(\mathcal{L}(M^2, S_0^2, F^2))$$

We define now the operator $post_M$ which computes with $post_M(B)(X)$ the set of states the system can access, starting from a state of X, after the occurrence of an event of B:

$$\begin{array}{rcccl} post_M : & \mathcal{P}(\Lambda) & \rightarrow & (\mathcal{P}(S) & \rightarrow & \mathcal{P}(S)) \\ & B & \mapsto & X & \mapsto & \cup\{\delta(\lambda, s) : \lambda \in B, s \in X\} \end{array}$$

The operator $reach_M$ computes with $reach_M(B)(X)$ the set of states reachable by a run starting in X and whose trace is a word over B. Formally, $reach_M(B)(X) = \cup\{(post_M(B))^i(X) : i \in \mathbb{N}\}$, so applying Proposition 2.2, we can define $reach_M$ by:

$$\begin{array}{rcccl} reach_M : & \mathcal{P}(\Lambda) & \rightarrow & (\mathcal{P}(S) & \rightarrow & \mathcal{P}(S)) \\ & B & \mapsto & X & \mapsto & lfp(Y \mapsto X \cup post_M(B)(Y)) \end{array}$$

For every $B \in \mathcal{P}(\Lambda)$, the operator $post_M(B)$ is monotone on the complete lattice $\mathcal{P}(S)$, so the operator $reach_M(B)$ is always defined according to Theorem 2.2.

In the thesis, we will often consider that only a subset Λ_o of the events of Λ can be observed when a run is executed. Then, we will note the projection $\pi_o = \pi_{\Lambda \rightarrow \Lambda_o}$ and the observed trace of a run is given by the observation map p_o defined by:

$$\begin{array}{rccl} p_o : & E(\Lambda, S) & \rightarrow & \Lambda_o^* \\ & r & \mapsto & \pi_o(tr(r)) \end{array}$$

In this context, the observable behavior, i.e. the set of observed traces, can be obtained by the ϵ-closure procedure. For this definition, let $\Lambda_{uo} = \Lambda \setminus \Lambda_o$.

Definition 2.18 (ϵ-closure) *If only the events of Λ_o are observable, the ϵ-closure of M is the LTS $\epsilon(M) = (\Lambda_o, S, \delta_\epsilon, X_0)$ where $X_0 = reach_M(\Lambda_{uo})(S_0)$ and*

$$\begin{array}{rccl} \delta_\epsilon : & \Lambda_o \times S & \rightarrow & S \\ & \lambda,\ s & \mapsto & reach_M(\Lambda_{uo}) \circ \delta(\lambda, s) \end{array}$$

3 Information Flow and Opacity

In this chapter, we introduce the notion of opacity. We consider a system given as an LTS $M = (\Lambda, S, \delta, S_0)$ where the sets Λ and S can possibly be infinite. This allows the following definitions to be general enough to apply our results to more specific models like process algebras, Petri nets, Timed Automata, etc.

Opacity is a concept of information flow and this notion depends on the assumptions that are made regarding the aptitudes of the attackers. We start this chapter by defining the kind of information that should remain confidential and by giving some hypothesis about the attackers that will hold for the rest of the thesis.

3.1 Confidential Information and Notion of Attackers

We start this section by discussing the kind of secret information that we expect to be concealed. This is formalized with the notion of secret predicate defined below. Then, we illustrate with some examples how the problem of inferring the truth of some secret predicates under partial observation arises in the context of security.

Definition 3.1 *A* secret predicate *ϕ is a predicate over $E(\Lambda, S)$ such that the occurrence of a run in M satisfying ϕ is an information that should remain confidential to unauthorized users.*

Example 3.1 *Let $Var = \{x_1, x_2, \ldots, x_m, k\}$ be a finite set of variables used in a program P implementing a cryptographic primitive. Let $Dom(x)$ denote the domain of a variable x. Suppose now that k is used to store the value of a cryptographic key. For such a system, it is required that an attacker cannot acquire more precise information about the value of k than its domain $Dom(k)$. The set $S = Dom(x_0) \times Dom(x_1) \times \cdots \times Dom(x_m) \times Dom(k)$ is the set of all possible configurations and Λ is a set of labels for the transitions. Let M denote the execution graph of the program. Let $v \in Dom(k)$ and consider $F_v^{\leq} = \{s \in S : value(k) \leq v\}$, $F_v^{>} = S \setminus F_v^{\leq}$. Consider the secret predicates $\phi_v^{\leq}$ and $\phi_v^{>}$ defined for $r \in E(\Lambda, S)$ by $r \models \phi_v^{\leq}$ when $lst(r) \in F_v^{\leq}$ and $r \models \phi_v^{>}$ when $lst(r) \in F_v^{>}$. Then, the attacker, partially observing the system, should not be able to infer that an execution $r \in \mathcal{R}(M)$ is such that $r \models \phi_v^{\leq}$ since otherwise, the attacker knows the information "$value(k) \leq v$". In this* *example, it is also required that the predicate $\phi_v^>$ does not leak to ensure the secrecy of $value(k)$.*

Let A denote an attacker whose aim is to infer that the current run executed by M satisfies ϕ. We understand A to be a machine, or a class of machines, and not a real person. In that case we can precisely define its capabilities and knowledge. By consequence, certifying that M preserves the confidentiality of ϕ depends on the considered model for the attacker.

We suppose that the attacker has an imperfect information about what really happens when a run is executed. This information is given by an *observation map* $obs : E(\Lambda, S) \to \mathcal{O}$ where $\mathcal{O}$ is a set of possible observations and $obs(r)$ is what the attacker observes when a run r is executed by M. For example, the attacker can observe the values of a subset of the set of variables. This defines a map $S \to V_o$ where V_o are the possible values of the observable variables. Then, two configurations of S have the same image when the values of the observable variables are the same. The resulting observation map is a map $obs : E(\Lambda, S) \to V_o^*$ where the image of a run r is the sequence of the values of the observed variables. Also, we can consider that a subset of the events are observable (e.g. the inputs and the outputs) and then, the partial observation of a run will only depend on its trace. Also, the observability of the events can dynamically evolve at runtime, as it will be the case in Chapter 6.

We can sometimes model a partial observation based on observable variables, as suggested above, in an event based fashion. For this reason, we will only consider in this thesis observation maps that are *trace-based*, i.e. defined from a map $\pi : \Lambda^* \to \mathcal{O}$ by $obs = \pi \circ tr$. But, to ease the presentation, we will make assumptions about obs only when necessary.

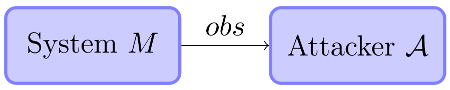


Figure 3.1: General architecture

The situation we consider is depicted in figure 3.1 where an inquisitive attacker wants to infer whenever ϕ is satisfied by the run currently executed by M. Then, the objective of the attacker is to compute a function $\gamma_\phi : \mathcal{R}(M) \to \{true, ?\}$ to decide the truth of ϕ. But the attacker only partially observes the executions of M and therefore only accesses to the outcomes of the map obs. In other words the function γ_ϕ must be such that there exists another function Γ_ϕ such that $\gamma_\phi = \Gamma_\phi \circ obs$. Finally, the objective of the attacker is to compute such a function Γ_ϕ, called a *monitor*, associating to each observation a verdict concerning the satisfaction of the predicate ϕ. This monitor must be sound in the sense that it must provide sound verdicts regarding the satisfaction of ϕ on the runs of M. This can be defined as follows

Definition 3.2 (Sound monitor) *A* sound monitor *for a predicate ϕ is a function $\Gamma_\phi : \mathcal{O} \to \{true, ?\}$ providing sound verdicts, i.e. for all run $r \in \mathcal{R}(M)$,*

$$\Gamma_\phi(obs(r)) = true \implies r \models \phi$$

Note that due to the partial observation, the other implication does not hold in general. For example, the monitor mapping every observation to ? is sound regardless of the possible satisfaction of ϕ by some runs of M.

We say that there is an information flow from the system to the attacker whenever for some sound monitor Γ_ϕ and some run $r \in \mathcal{R}(M)$, $\Gamma_\phi(obs(r)) = true$.

Example 3.2 *We can model the behavior of the code of Example 1.1, by its control flow graph $M = (\Lambda, S, \delta, S_0)$ where:*

- *the set of events is $\Lambda = \Lambda_a \cup \Lambda_{ua}$ with $\Lambda_a = \{$`receive(m, A)`$: m \in \mathbb{N}\} \cup \{$`send(m, A)`$: m \in \mathbb{N}\}$ are observable events and $\Lambda_{ua} = \{\tau,$ `k = random()`, `x = random()`$\}$ are not observable;*
- *the set of states $S = \mathbb{N} \times \mathbb{N}$ and $S_0 = \{(0, 0)\}$;*

The observation $obs : E(\Lambda, S) \to \Lambda_a^*$ maps every run r to the sequence of `receive(m, A)` *and* `send(m, A)`*, $m \in \mathbb{N}$, occurring in this run. To formalize the situation presented on the example 1.1, we can define the secret predicate ϕ by: $r \models \phi$ when $lst(r) \in \{6\} \times \mathbb{N}$. The monitor for ϕ suggested in Example 1.1 is implemented by the automaton pictured in the figure 3.2. This monitor is sound since the event* `send(0, A)` *is possible only when the*

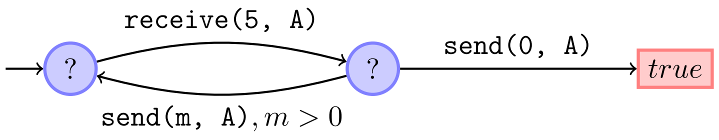


Figure 3.2: Example of sound monitor

current run of M satisfies ϕ.

3.2 Definition of Opacity

We now make the additional assumption that the attackers may have a complete knowledge about the semantics of M, i.e. the set of runs $\mathcal{R}(M)$. We define now the notion of *opacity* which is based on this assumption. Intuitively, a secret predicate ϕ is opaque if no attacker, even the ones with full knowledge of $\mathcal{R}(M)$, can ever infer from what is observed that the current run of M satisfies ϕ.

Definition 3.3 (Opacity [BKMR08]) *A secret predicate ϕ is opaque on M for obs if for all $r \in \mathcal{R}(M)$ such that $r \models \phi$, there exists $r' \in \mathcal{R}(M)$ such that $obs(r) = obs(r')$ and $r' \not\models \phi$*

We will also say that M is ϕ-opaque for *obs* when the opacity of ϕ is satisfied. If M is ϕ-opaque for *obs*, then no attacker observing the outcomes of *obs* can infer whether ϕ is satisfied on the runs of M. In other words, it is impossible to compute a sound monitor Γ_ϕ such that $\Gamma_\phi(\nu) = true$ for some $\nu \in obs(\mathcal{R}(M))$.

Theorem 3.1 *The LTS M is ϕ-opaque for obs if and only if for every sound monitor Γ_ϕ and every run $r \in \mathcal{R}(M)$, $\Gamma_\phi(obs(r)) \neq true$.*

Proof. Suppose that M is ϕ-opaque for *obs* and let $r \in \mathcal{R}(M)$. There are two possibilities:

1. if $r \not\models \phi$, then $\Gamma_\phi(obs(r)) \neq true$ since Γ_ϕ is sound;
2. if $r \models \phi$, then M being ϕ-opaque for *obs*, there exists another run r' such that $obs(r') = obs(r)$ and $r' \not\models \phi$. Since Γ_ϕ is sound, we must have $\Gamma_\phi(obs(r')) = ?$. So, $obs(r) = obs(r')$ implies $\Gamma_\phi(obs(r)) = \Gamma_\phi(obs(r')) \neq true$.

Finally, $\Gamma_\phi(obs(r)) \neq true$.

For the other implication, an attacker with full knowledge of $\mathcal{R}(M)$ can define the following monitor:

$$\begin{array}{rccl} \Gamma_\phi : & \mathcal{O} & \to & \{true, ?\} \\ & \nu & \mapsto & \begin{cases} true \text{ when } \forall r \in obs^{-1}(\nu) \cap \mathcal{R}(M),\ r \models \phi \\ ? \text{ otherwise} \end{cases} \end{array}$$

So if M is not ϕ-opaque for *obs* then there exists $r \in \mathcal{R}(M)$ such that $\forall r' \in obs^{-1}(obs(r)) \cap \mathcal{R}(M)$, $r' \models \phi$. Then $\Gamma_\phi(obs(r)) = true$. □

Opacity is then a necessary and sufficient condition to avoid that an attacker knowing $\mathcal{R}(M)$ can infer the truth of ϕ using sound monitors.

Example 3.3 *Consider the LTS pictured in figure 3.3 where the attacker observes the events a and b but not τ. The secret ϕ is satisfied on the runs ending in the square states s_3 or s_5. If the run $r = s_0 \xrightarrow{\tau} s_2 \xrightarrow{b} s_5$ is executed, then, knowing the model M, the attacker knows that the only possible runs explaining the observation b is r, ending in s_5. Then, the information "M is currently in a secret state" is disclosed for the observed trace b. Note that if the attacker observes the trace a, the attacker cannot distinguish whether $s_0 \xrightarrow{\tau} s_1 \xrightarrow{a} s_3$ or $s_0 \xrightarrow{\tau} s_2 \xrightarrow{a} s_4$ has been executed and cannot infer that the current state is s_3 or s_4 and no secret information is disclosed.*

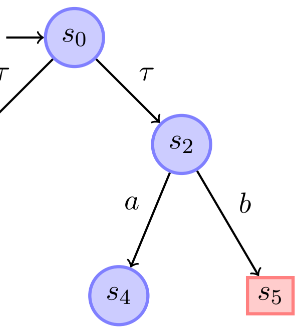


Figure 3.3: Simple example of non opacity

Next, we characterize the set of counterexamples to the opacity of ϕ, i.e. the set of runs of M such that the secret predicate is disclosed.

Definition 3.4 *The set of runs such that the secret predicate ϕ is disclosed is denoted:*

$$Disclose(\mathcal{R}(M), obs)(\phi) = \{r \in \mathcal{R}(M) : \forall r' \in obs^{-1}(obs(r)) \cap \mathcal{R}(M),\ r' \models \phi\}$$

This definition provides another formulation of the ϕ-opacity of M as the absence of disclosing run:

Proposition 3.1 *M is ϕ-opaque for obs $\iff$ $Disclose(\mathcal{R}(M), obs)(\phi) = \emptyset$.*

For example, considering M and ϕ from the LTS of Example 3.3, $Disclose(M, obs)(\phi) = \{s_0 \xrightarrow{\tau} s_2 \xrightarrow{b} s_5\}$ and M is therefore not ϕ-opaque for an attacker observing a and b.

We also formulate the set of counterexamples to opacity as follows. Denoting by R_ϕ the elements of $E(\Lambda, S)$ satisfying ϕ, the set $obs^{-1}(obs(\mathcal{R}(M) \setminus R_\phi))$ is the set of runs (of $E(\Lambda, S)$) such that there exists at least one run of M with the same observation and not satisfying ϕ. So removing this set from $\mathcal{R}(M)$, we obtain:

$$Disclose(\mathcal{R}(M), obs)(\phi) = \mathcal{R}(M) \setminus obs^{-1}(obs(\mathcal{R}(M) \setminus R_\phi))$$

We characterize now the set of observed traces on the basis of which an attacker knowing the model M can infer the truth of ϕ.

Definition 3.5 *The set of observations such that the secret ϕ is disclosed is the set*

$$DTraces(\mathcal{R}(M), obs)(\phi) = \{obs(r) \in \mathcal{O} : r \in Disclose(\mathcal{R}(M), obs)(\phi)\}$$

The operator $DTraces(\mathcal{R}(M), obs)(\cdot)$ will be useful in the sequel to prove that a model transformation preserves the opacity property by stating that the set of observed traces such that secret information is disclosed is an invariant of the transformation.

We can see on the example 3.1, that the program is secure whenever both $\phi_v^{\leq}$ and $\phi_v^{>}$ are opaque. More generally, it is common when confidentiality is required that several, possibly interdependent, secret predicates have to be simultaneously opaque. To take this aspect into account, the notion of opacity is generalized to several secret predicates.

Definition 3.6 (Multi-secret opacity) *Given a finite set of secret predicates Φ. We say that M is Φ-opaque for obs when for every secret predicate $\phi \in \Phi$, M is ϕ-opaque for obs.*

Given a set of secret predicates Φ, the set of runs such that at least one secret is disclosed is then given by:

$$Disclose(\mathcal{R}(M), obs)(\Phi) = \cup\{Disclose(\mathcal{R}(M), obs)(\phi) : \phi \in \Phi\}$$

For example, as suggested by the previous example, the notion of opacity can then be used to define the notion of *secrecy* like in [AČZ06, TO08, Mas08, DGMD06], where one considers the system to be unsafe when an attacker either knows that a confidential predicate is satisfied or knows that the predicate is not satisfied. This notion of secrecy is especially suitable to reason about the confidentiality of variables, as in Example 3.1. In our context, the notion of secrecy of a predicate ϕ can be expressed as the conjoint opacity of ϕ and $\neg\phi$.

For a finite set of secret predicates Φ, the objective of an attacker with full knowledge of the set $\mathcal{R}(M)$ is to compute the canonical monitor defined as follows.

Definition 3.7 *Given a finite set Φ of secret predicates, the canonical monitor for Φ is*

$$\begin{array}{rccl} \Gamma_\Phi : & \mathcal{O} & \rightarrow & \mathcal{P}(\Phi) \\ & \nu & \mapsto & \{\phi \in \Phi : \nu \in DTraces(\mathcal{R}(M), obs)(\phi)\} \end{array}$$

Note that this definition of monitor does not correspond to the one introduced earlier where the codomain was the set $\{true, ?\}$. In this case, as there is several predicates, we replace the verdict *true* by the set of secret predicates whose truth can be inferred for the given observed trace.

3.3 Properties of Opacity

We will study in this section some general aspects of the notion of opacity. We start this section with some results that will be useful in the subsequent chapters.

3.3.1 Some General Properties of Opacity

First, we state that if a predicate ϕ is more general than a predicate ϕ', then an attacker can infer truth of ϕ by inferring the truth of ϕ'. This can be useful, for example when the predicate ϕ' is an underapproximation of ϕ, then an attacker can state sound verdicts about ϕ by considering ϕ' instead of ϕ.

Proposition 3.2 *If $\phi' \Rightarrow \phi$, then $\mathit{Disclose}(\mathcal{R}(M), \mathit{obs})(\phi') \subseteq \mathit{Disclose}(\mathcal{R}(M), \mathit{obs})(\phi)$.*

The next proposition formalizes that the most accurate the attacker is, the most likely is the occurrence of information flow.

Proposition 3.3 *If obs_1 and obs_2 are two observation maps such that for all $r, r' \in \mathcal{R}(M)$, $\mathit{obs}_1(r) = \mathit{obs}_1(r')$ implies that $\mathit{obs}_2(r) = \mathit{obs}_2(r')$, then*

$$\mathit{Disclose}(\mathcal{R}(M), \mathit{obs}_2)(\phi) \subseteq \mathit{Disclose}(\mathcal{R}(M), \mathit{obs}_1)(\phi)$$

Proof. Let $r \in \mathit{Disclose}(\mathcal{R}(M), \mathit{obs}_2)(\phi)$ and $r' \in \mathcal{R}(M)$ such that $\mathit{obs}_1(r') = \mathit{obs}_1(r)$. Then $\mathit{obs}_2(r') = \mathit{obs}_2(r)$ and we know then that $r' \models \phi$ since $r \in \mathit{Disclose}(\mathcal{R}(M), \mathit{obs}_2)(\phi)$. So $r \in \mathit{Disclose}(\mathcal{R}(M), \mathit{obs}_1)(\phi)$. $\square$

We will study now what are the effects of inclusion on the opacity property. We consider systems represented by their semantics, i.e. their sets of runs. The next proposition will have important consequences in Chapter 4 when we consider approximation techniques to analyze opacity.

Proposition 3.4 *Let R_1 and R_2 be two systems such that $R_1 \subseteq R_2 \subseteq E(\Lambda, S)$ and ϕ be a secret predicate. Then,*

$$R_1 \cap \mathit{Disclose}(R_2, \mathit{obs})(\phi) \subseteq \mathit{Disclose}(R_1, \mathit{obs})(\phi)$$

Proof. Let $r \in R_1 \cap \mathit{Disclose}(R_2, \mathit{obs})(\phi)$. Let $r' \in R_1$ such that $\mathit{obs}(r) = \mathit{obs}(r')$. Then, r' is also in R_2 and as $r \in \mathit{Disclose}(R_2, \mathit{obs})(\phi)$, $r' \models \phi$. Finally $r \in \mathit{Disclose}(R_1, \mathit{obs})(\phi)$. $\square$

3.3.2 Trace-based Observation Maps

One can imagine that the secret predicate ϕ is a temporal logic formula interpreted over the runs of M where the valuation function for the atomic propositions is either based on the set of states or on the set of events. For the techniques developed in this thesis to be general enough, we consider two kinds of secret predicates: state-based and trace-based ones, defined below.

Definition 3.8 (State-based predicate) *We say that the predicates of Φ are state-based if they are defined by the reachability of certain states of S, i.e. there exists a map $F : \Phi \to \mathcal{P}(S)$, $\phi \mapsto F(\phi)$ such that for all $r \in E(\Lambda, S)$, $r \models \phi$ whenever $lst(r) \in F(\phi)$.*

For example, the state-based predicate of Example 3.3 is defined by $F(\phi) = \{s_3, s_5\}$.

Definition 3.9 (Trace-based predicate) *We say that the predicates of Φ are trace-based if there exists a map $L : \Phi \to \mathcal{P}(\Lambda^*), \phi \mapsto L(\phi)$ such that for all $r \in E(\Lambda, S)$, $r \models \phi$ whenever $tr(r) \in L(\phi)$.*

Example 3.4 *Consider the deterministic LTS pictured in Figure 3.4, representing the behavior of a vending machine for tea and coffee. A thief interested in stealing the money contained in the machine wants to take the risk to break the machine only when it is certain that the money box is full of cash. But the thief does not directly observe the money box. This secret information can be modeled by the trace-based predicate ϕ defined by $L(\phi) = \Sigma^* cashFull \Sigma^*$.*

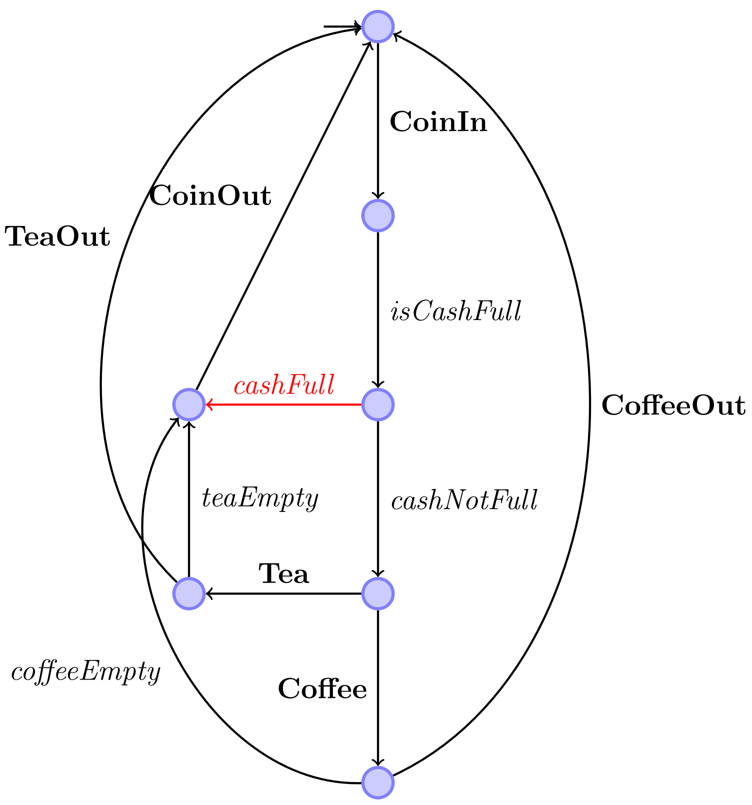


Figure 3.4: An insecure vending machine

Definition 3.10 (Regular predicate) *When the alphabet of events Λ of M is finite, a trace-based predicate ϕ such that $L(\phi)$ is a regular language will be called a regular secret predicate.*

In the next chapters, we will make the assumption that the observation map *obs* is trace-based, i.e. there exists a map $\pi : \Lambda^* \to \mathcal{O}$ such that $obs = \pi \circ tr$. We give now some useful results that are consequences of this assumption about *obs*.

Proposition 3.5 *Let M and M' be two LTSs such that $\mathcal{L}(M) = \mathcal{L}(M')$. If ϕ is a trace based predicate, then $DTraces(\mathcal{R}(M), obs)(\phi) = DTraces(\mathcal{R}(M'), obs)(\phi)$.*

Proof. Let $\nu \in DTraces(\mathcal{R}(M), obs)(\phi)$ and $r' \in obs^{-1}(\nu) \cap \mathcal{R}(M')$. As $\mathcal{L}(M) = \mathcal{L}(M')$, there exists $r \in \mathcal{R}(M)$ such that $tr(r) = tr(r')$. But in that case, $obs(r) = \pi \circ tr(r) = \pi \circ tr(r') = obs(r')$. As $obs(r) = \nu$ and $\nu \in DTraces(\mathcal{R}(M), obs)(\phi)$, $tr(r) \in L(\phi)$. Finally, $tr(r') \in L(\phi)$ and $\nu \in DTraces(M', obs)(\phi)$. The other inclusion holds by symmetry. $\square$

In this context, we will use a language based approach for opacity problems. In order to ease the presentation, especially in Section 4.4 and Chapter 5, we will denote the set of counterexamples to ϕ-opacity by $Disclose(\mathcal{L}(M), \pi)(\phi)$ instead of $Disclose(\mathcal{R}(M), obs)(\phi)$. Similarly, we will denote by $DTraces(\mathcal{L}(M), \pi)(\phi)$ the set of observed traces disclosing the secret ϕ.

We will see that, with this assumption on *obs*, an opacity problem with trace-based predicates can also be encoded by an opacity problem with states-based ones.

Let $\Phi = \{\phi_1, \phi_2, \dots, \phi_k\}$ be a set of trace-based predicates over $E(\Lambda, S)$. Suppose that for each $i \in \{1, 2, \dots, k\}$, there exists a complete and deterministic LTS $A_i = (\Lambda, Q^i, \delta^i, q_0^i)$ with $Q_f^i \subseteq Q^i$ such that $L(\phi_i) = \mathcal{L}(A_i, q_0^i, Q_f^i)$.

Remark 3.1 *Note that such an LTS A_i always exists. Indeed, a possibility is to let $Q^i = \Lambda^*$, $q_0^i = \epsilon$, $\delta^i(\lambda, w) = w\lambda$ and $Q_f^i = L(\phi_i)$, and complement the resulting LTS. But, in the subsequent chapters, we will use the following results when ϕ_i is a regular predicate usually given as a complete DFA A_i.*

Define now $S_\Phi = S \times Q^1 \times Q^2 \times \dots \times Q^k$ and $M_\Phi = M \parallel A_1 \parallel A_2 \parallel \dots \parallel A_k$. Given a run $r \in \mathcal{R}(M)$,

$$r = s_0 \xrightarrow{\lambda_1} s_1 \xrightarrow{\lambda_2} s_2 \ \cdots \ s_{n-1} \xrightarrow{\lambda_n} s_n$$

there exists then a unique run $\tilde{r} \in \mathcal{R}(M_\Phi)$:

$$\tilde{r} = (s_0, q_0^1, \dots, q_0^k) \xrightarrow{\lambda_1} (s_1, q_1^1, \dots, q_1^k) \ \cdots \ \xrightarrow{\lambda_n} (s_n, q_n^1, \dots, q_n^k)$$

We define the state-based predicates $\tilde{\Phi} = \{\tilde{\phi}_i : 1 \leq i \leq k\}$ by $F(\tilde{\phi}_i) = \{(s, q^1, q^2, \dots, q^k) \in S_\Phi,\ q^i \in Q_f^i\}$, and we obtain the following proposition:

Proposition 3.6 *Let $\phi \in \Phi$. For all $r \in \mathcal{R}(M)$, $r \models \phi$ if and only if $\tilde{r} \models \tilde{\phi}$*

Proof. Follows directly the construction of M_Φ where every A_i is complete and deterministic. □

Proposition 3.7 *For all $\phi \in \Phi$, $DTraces(\mathcal{R}(M), obs)(\phi) = DTraces(\mathcal{R}(M_\Phi), obs)(\tilde{\phi})$.*

Proof. Let $\nu \in obs(\mathcal{R}(M))$. First, we observe that

$$r \in obs^{-1}(\nu) \cap \mathcal{R}(M) \iff \tilde{r} \in obs^{-1}(\nu) \cap \mathcal{R}(M_\Phi)$$

Then,

$$\begin{aligned} \nu \in DTraces(\mathcal{R}(M), obs)(\phi) &\iff \forall r \in obs^{-1}(\nu) \cap \mathcal{R}(M),\ tr(r) \in L(\phi) \\ &\iff \forall \tilde{r} \in obs^{-1}(\nu)\mathcal{R}(M_\Phi),\ lst(\tilde{r}) \in F(\phi) \\ &\iff \nu \in DTraces(M_\Phi, obs)(\tilde{\phi}) \end{aligned}$$

□

Then, the problem of verifying opacity for the trace-based predicate ϕ on M becomes equivalent to the problem of verifying opacity for the state-based predicate $\tilde{\phi}$ on M_Φ.

The encoding can also be defined in the other direction. Let ϕ be a state-based secret predicate. We define the trace-based predicate ϕ_t by $L(\phi_t) = \mathcal{L}(M, S_0, F(\phi)) \setminus \mathcal{L}(M, S_0, S \setminus F(\phi))$.

Proposition 3.8 *$DTraces(\mathcal{R}(M), obs)(\phi) = DTraces(\mathcal{R}(M), obs)(\phi_t)$*

Proof. First, we note that for $r \in \mathcal{R}(M)$, $tr(r) \in L(\phi_t)$ if and only if for all $r' \in \mathcal{R}(M)$ such that $tr(r') = tr(r)$, $lst(r') \in F(\phi)$. Now let $\nu \in DTraces(\mathcal{R}(M), obs)(\phi)$. Let $r \in obs^{-1}(\nu) \cap \mathcal{R}(M)$ and $r' \in \mathcal{R}(M)$ such that $tr(r') = tr(r)$. Then, $obs(r') = \pi \circ tr(r') = obs(r) = \nu$ and so $lst(r') \in F(\phi)$. So, $tr(r) \in L(\phi_t)$ and $r \models \phi_t$. The other implication follows directly the definition of $L(\phi_t)$. □

3.4 Conclusion

To summarize this chapter, we have presented how the problem of inferring the truth of a predicate over the runs arises in security. Then we have defined the notion of opacity, formalizing the impossibility of an attacker to infer such an information on the basis of sound monitors. We have also characterized the set of counterexamples to opacity, with the operator *Disclose* and the associated set of observation disclosing the truth of a secret predicate with the operator *Dtraces*. Based on this operator *DTraces*, we have shown that we can choose the kind of secret predicate, state-based or trace-based, that is the most suitable to investigate a problem related to opacity.

In the next chapter, we will investigate the opacity verification problem for finite and infinite systems. For this, we will follow two approaches, applying abstract interpretation techniques and adapt the notion of diagnosis to the detection of information flow.

4 Verifying and Monitoring Opacity

In this chapter, we consider an LTS $M = (\Lambda, S, \delta, S_0)$ and a finite set of state-based secret predicates $\mathit{\Phi}$. The objective of an attacker is to observe traces of $DTraces(\mathcal{R}(M), obs)(\phi)$ to infer the truth of a predicate $\phi \in \mathit{\Phi}$. For this purpose, we will see how to construct the sound monitors introduced in Definition 3.7 for the predicates $\mathit{\Phi}$.

We make some assumptions about the observation map that will hold for the whole chapter: we assume that the attacker only observes a subset Λ_a of the events of M and we consider the corresponding map: $\pi_a = \pi_{\Lambda \to \Lambda_a} : \Lambda^* \to \Lambda_a^*$ as introduced in Chapter 2, and $p_a : E(\Lambda, S) \to \Lambda_a^*$ defined by $p_a = \pi_a \circ tr$. We denote $\Lambda_{ua} = \Lambda \setminus \Lambda_a$ the set of unobservable events.

First, we will investigate how to express $DTraces(\mathcal{R}(M), p_a)(\mathit{\Phi})$ and the corresponding monitor in a general setting, without constraints about the effective computability of the needed operations like state reachability or determinization. In this chapter, we will first investigate how an abstract interpretation framework based on Galois connections, like in [CC77b, CC92a, GM04], can be applied to approximate the determinization of M. We also study how to detect the existence of counterexample to opacity by considering a regular abstraction of M and applying the diagnosis theory.

4.1 Determinization Based Procedure to Construct Sound Monitors

We investigate in this section a method to exhibit attack scenarios, i.e. elements of the set of observations $DTraces(\mathcal{R}(M), p_a)(\phi)$, for $\phi \in \mathit{\Phi}$. The problem we search to solve is the following.

Problem 4.1

- ***Input:*** *A possibility infinite LTS $M = (\Lambda, S, \delta, S_0)$, a finite set of secret predicates $\mathit{\Phi}$ and an attacker observing M through the projection $p_a : E(\Lambda, S) \to \Lambda_a^*$.*
- ***Problems:***
  - ***(A)*** *Decide whether a trace $\nu \in p_a(\mathcal{R}(M))$ observed from M belongs to $DTraces(\mathcal{R}(M), p_a)(\phi)$ for some $\phi \in \mathit{\Phi}$, i.e. discloses the truth of ϕ.*

– ***(B)*** *Decide whether $DTraces(\mathcal{R}(M), p_a)(\Phi) = \emptyset$, i.e. whether M is Φ-opaque for p_a.*

To solve Problem 4.1, we propose a method based on the operators $post_M$ and $reach_M$ defined at the end of chapter 2. As there is no ambiguity in this chapter about the system M, we will note *post* and *reach* instead of $post_M$ and $reach_M$ to simplify the notations. We also assume that for all $B \subseteq \Lambda$, the operator $post(B)$ is ω-continuous. In the sequel, we will use the notation $reach_{ua} : \mathcal{P}(S) \to \mathcal{P}(S)$ for $reach(\Lambda_{ua})$ (recall that $\Lambda_{ua} = \Lambda \setminus \Lambda_a$).

Remark 4.1 *The effective computation of reach may be impossible since the fixpoint computation may not terminate. We will see in Section 4.3 how to circumvent this problem by approximating the operator.*

In order to compute $DTraces(\mathcal{R}(M), p)(\phi)$ for every $\phi \in \Phi$, the first step consists in computing the set of states that M may have reached when a trace is observed. This is done by the subset construction for determinization defined below.

Definition 4.1 *The determinization of M with respect to Λ_a gives the deterministic LTS $det_a(M) = (\Lambda_a, \mathcal{P}(S), \Delta_a, X_0)$ where the initial state $X_0 = reach_{ua}(S_0)$ is the set of states that are reachable when the attacker observes nothing (i.e. the empty trace ϵ) and the transition function is defined by:*

$$\begin{array}{rlcl} \Delta_a : & \Lambda_a \times \mathcal{P}(S) & \to & \mathcal{P}(S) \\ & \lambda,\ X & \mapsto & reach_{ua} \circ post(\{\lambda\})(X) \ \textit{when not empty} \end{array}$$

The transition function Δ_a of the deterministic LTS $det_a(M)$ is extended to words according to the definition of Section 2.2.

Example 4.1 *Consider that M is the LTS of example 3.3 with $\Lambda_a = \{a, b\}$. Then, $det_a(M)$ is the LTS depicted in figure 4.1.*

$$\begin{tikzcd}
{} \arrow[r] & \{s_0, s_1, s_2\} \arrow[dl, "a"'] \arrow[dr, "b"] & \\
\{s_3, s_4\} & & \{s_5\}
\end{tikzcd}$$

Figure 4.1: Example of determinization

The next proposition provides a useful characterization of the outcomes of the map $\Delta_a(\,\cdot\,, X_0) : \Lambda_a^* \to \mathcal{P}(S)$. Informally, $\Delta_a(\nu, X_0)$ is the set of states the system may have reached when the attacker observes ν.

Proposition 4.1 *For all $\nu \in \Lambda_a^*$,*

$$\Delta_a(\nu, X_0) = \{s \in S : \exists r \in \mathcal{R}(M),\ p_a(r) = \nu \wedge lst(r) = s\}$$

Proof. For $\nu \in \Lambda_a^*$, let $Z(\nu) = \{s \in S : \exists r \in \mathcal{R}(M),\ p_a(r) = \nu \ \wedge \ lst(r) = s\}$. It is clear from the definition of Δ_a that for all $\nu \in \Lambda_a^*$, $\Delta_a(\nu, X_0) \subseteq Z(\nu)$. Let $n \in \mathbb{N}$ and suppose that for all $\nu \in \Lambda_a^n$, $\Delta_a(\nu, X_0) = Z(\nu)$. Now, let $\nu\lambda \in \Lambda_a^{n+1}$ and $s \in Z(\nu\lambda)$. There exists $r \in R(M)$ such that $p_a(r) = \nu\lambda$ and $lst(r) = s$. Let r_1 be the longest prefix of r such that $p_a(r_1) = \nu$. Then we can write $r = r_1 \cdot r_2$ with $r_2 = s_1 \overset{\lambda}{\to} s_2 \overset{u}{\to} s$, $u \in \Lambda_{ua}^*$ and $s_1 \in Z(\nu)$. So $s \in \Delta_a(\lambda, \{s_1\}) \subseteq \Delta_a(\lambda, Z(\nu))$. Since $|\nu| = n$, $\Delta_a(\nu, X_0) = Z(\nu)$ so $s \in \Delta_a(\lambda, \Delta_a(\nu, X_0)) = \Delta_a(\nu\lambda, X_0)$. Finally $\Delta_a(\nu\lambda, X_0) = Z(\nu\lambda)$. The hypothesis also holds for $n = 0$ since $Z(\epsilon) = reach_{ua}(S_0) = X_0 = \Delta_a(\epsilon, X_0)$ which proves the proposition by induction. □

Corollary 4.1 *If $r \in \mathcal{R}(M)$, then $lst(r) \in \Delta_a(p_a(r), X_0)$.*

Using Proposition 4.1 and Corollary 4.1, it follows that:

Corollary 4.2 *$\mathcal{L}(det_a(M)) = p_a(\mathcal{R}(M))$.*

Next, we relate the set of observations such that the secret predicate is disclosed to the reachability of certain states in the LTS $det_a(M)$.

Proposition 4.2 *$DTraces(\mathcal{R}(M), p_a)(\phi) = \mathcal{L}(det_a(M), X_0, \mathcal{P}(F(\phi)))$ for every $\phi \in \Phi$.*

Proof. Let $\phi \in \Phi$. Let $\nu \in \mathcal{L}(det_a(M), X_0, \mathcal{P}(F(\phi)))$. According to Corollary 4.1, for all $r \in p_a^{-1}(\nu) \cap \mathcal{R}(M)$, $lst(r) \in \Delta_a(\nu, X_0) \subseteq F(\phi)$. This implies that $\nu \in DTraces(\mathcal{R}(M), p_a)(\phi)$. For the other inclusion, let $\nu \in DTraces(\mathcal{R}(M), p_a)(\phi)$. Then, there exists $r \in \mathcal{R}(M)$ such that $p_a(r) = \nu$ so $\nu \in \mathcal{L}(det_a(M))$ according to Corollary 4.2. Let $s \in \Delta_a(\nu, X_0)$. According to Proposition 4.1, there exists $r' \in \mathcal{R}(M)$ such that $p_a(r') = \nu$ and $s = lst(r')$. But, since $\nu \in DTraces(\mathcal{R}(M), p_a)(\phi)$, we know that $r' \models \phi$ i.e. $s = lst(r') \in F(\phi)$. Then $\Delta_a(\nu, X_0) \subseteq F(\phi)$ and $\nu \in \mathcal{L}(det_a(M), X_0, \mathcal{P}(F(\phi)))$. □

Those two propositions imply the following theorem which also implies the existence of an algorithm for verifying opacity based on determinization and reachability analysis.

Theorem 4.1 *The system M is Φ-opaque for p_a if and only if for all $\phi \in \Phi$, the states $\mathcal{P}(F(\phi))$ are not reachable in $det_a(M)$.*

Proof. Follows from proposition 3.1 and 4.2. □

This theorem also allows to compute the canonical monitor from Definition 3.7 as follows.

Proposition 4.3 *Given a finite set Φ of secret predicates, the canonical monitor for Φ is then given by:*

$$\begin{array}{rrcl} \Gamma_\Phi : & \Lambda_a^* & \to & \mathcal{P}(\Phi) \\ & \nu & \mapsto & \{\phi \in \Phi : \Delta_a(\nu, X_0) \subseteq F(\phi)\} \end{array}$$

4.2 Complexity of Verifying Opacity on Finite Models

If Λ and S are finite sets, $det_a(M)$ can always be computed. Then, $DTraces(\mathcal{R}(M), p_a)(\phi)$ can be computed for every secret predicate $\phi \in \Phi$ as well as the canonical monitor Γ_Φ. As a consequence, the problems 4.1 (A) and (B) are decidable for finite LTSs. More precisely, we will see in this section that the opacity verification problem is PSPACE-complete. To prove this, we will prove that the universality problem for NFA can be encoded as two opacity verification problems and that a procedure to verify universality can be used to verify opacity.

Let $A = (\Sigma, Q, \delta_A, Q_0, Q_f)$ be a finite and possibly non-deterministic automaton. We say that A is language universal for Q_f when $\mathcal{L}(A, Q_0, Q_f) = \Sigma^*$. Deciding language universality on an automaton A is known to be complete for PSPACE [SM73].

Suppose now that the attacker observes all the events of Σ. Then, the corresponding observation map is the trace operator $tr : E(\Sigma, Q) \to \Sigma^*$. We define the state-based secret predicate ϕ_f over $E(\Sigma, Q)$ by $F(\phi_f) = Q \setminus Q_f$. Also, consider the LTS $comp(A) = (\Sigma, Q \cup \{\tilde{q}\}, \delta_A^c, Q_0)$ where $\tilde{q} \notin Q$ is the new added state following the construction of a complete LTS of Definition 2.16. Define also the secret predicate $\tilde{\phi}$ over $E(\Sigma, Q \cup \{\tilde{q}\})$ by $F(\tilde{\phi}) = \{\tilde{q}\}$. With this construction, we obtain

Proposition 4.4 *A is language universal for Q_f if and only if A is ϕ-opaque for tr and $comp(A)$ is $\tilde{\phi}$-opaque for tr.*

Proof.

$$\begin{array}{rcl} A \text{ universal} & \iff & \mathcal{L}(A) = \Sigma^* \text{ and } \mathcal{L}(A, Q_0, Q_f) = \mathcal{L}(A) \\ & \iff & \mathcal{L}(comp(A), Q_0, Q) = \mathcal{L}(comp(A)) \text{ and } \mathcal{L}(A, Q_0, Q_f) = \mathcal{L}(A) \\ & \iff & \forall \tilde{\rho} \in \mathcal{R}(comp(A)),\ \exists \tilde{\rho'} \in \mathcal{R}(comp(A), Q_0, Q),\ tr(\tilde{\rho}) = tr(\tilde{\rho'}) \\ & & \text{and } \forall \rho_f \in \mathcal{R}(A),\ \exists \rho'_f \in \mathcal{R}(A, Q_0, Q_f),\ tr(\rho_f) = tr(\rho'_f) \\ & \iff & comp(A) \text{ is } \tilde{\phi}\text{-opaque for } tr \text{ and } A \text{ is } \phi_f\text{-opaque for } tr \end{array}$$

□

Consider now given a procedure for solving universality problems. We will show now that such a procedure can also be applied to verify opacity. Let $\phi \in \Phi$ with $F(\phi)$ the corresponding set of accepting states. Let $A = (\Lambda_a, S \cup \{q\}, \delta_A, X_0) = comp(\epsilon(M))$ following the notation of the definitions 2.16 and 2.18. Let the predicate ϕ_A be defined by $F(\phi_A) = \{q\} \cup F(\phi)$.

Lemma 4.1 *$DTraces(\mathcal{R}(M), p_a)(\phi) \subseteq DTraces(\mathcal{R}(A), tr)(\phi_A)$.*

Proof. Let $\nu \in DTraces(\mathcal{R}(M), p_a)(\phi)$ and $\rho \in \mathcal{R}(A)$ such that $tr(\rho) = \nu$. If $lst(\rho) = q$, then $r \models \phi_A$. Suppose now that $lst(\rho) \in S$. Then, there exists a run $r \in \mathcal{R}(M)$ such that $p_a(r) = tr(\rho)$ and $lst(r) = lst(\rho)$. Then, $p_a(r) = \nu$ so $lst(r) \in F(\phi) \subseteq F(\phi_A)$ and then $lst(\rho) \in F(\phi_A)$. Finally, $\nu \in DTraces(A, tr)(\phi_A)$. □

Proposition 4.5 *A is language universal for $S \setminus F(\phi)$ implies that M is ϕ-opaque for p_a.*

Proof. As $\mathcal{L}(A) = \Lambda_a^*$, applying Proposition 4.4, A is language universal for $S \cup \{q\} \setminus F(\phi_A)$ if and only $DTraces(A, tr)(\phi_A) = \emptyset$. As $S \setminus F(\phi) = S \cup \{q\} \setminus F(\phi_A)$, applying Lemma 4.1, the universality of A for $S \setminus F(\phi)$ implies that $DTraces(\mathcal{R}(M), p_a)(\phi) = \emptyset$, i.e. that M is ϕ-opaque for p_a. □

Theorem 4.2 *The problem 4.1 (B), i.e. the opacity verification problem, is PSPACE-complete.*

Proof. Assuming given a procedure to verify language universality, the computation of $\epsilon(M)$ from M and then the computation of $A = comp(\epsilon(M))$ can be done with a complexity polynomial in the size of M. Applying the proposition 4.5 and repeating the operation for each secret predicate $\phi \in \Phi$, we can encode the opacity problem as $|\Phi|$ universality problems which proves that the problem 4.1 is in PSPACE. Finally, the proposition 4.4 proves its PSPACE-completeness. Indeed, as a procedure to solve the opacity verification problem can be apply to solve universality, the opacity verification problem is therefore PSPACE-hard. □

Corollary 4.3 *Let M be a finite LTS and Φ be a finite set of regular trace-based secret predicates, then the opacity verification problem is PSPACE-complete.*

Proof. Given $\phi \in \Phi$, the product operation needed in Proposition 3.7 to obtain an equivalent state-based predicate can be done with a polynomial complexity in the size of M and the automaton accepting $L(\phi)$. □

Remark 4.2 *We have proposed an algorithm based on LTS determinization. But with the link between opacity and the universality problem described above, we can expect that a complete determinization procedure can be avoided, in some favorable situations, for example by applying antichains techniques developed in [DDHR06, DDR06] for solving the universality problem and therefore obtaining a more efficient algorithm for the opacity verification problem on finite models.*

4.3 Monitoring Opacity Using Abstract Interpretation

We consider now that the sets Λ and S are not necessarily finite. In that case, the fixed point computations required for the map Δ_a may not terminate. Then, the determinization procedure used to verify opacity on finite models cannot be directly applied. Furthermore, a reachability problem can easily be encoded as an opacity problem so it is clear that opacity is not decidable for Turing machines [BKMR08]. In the same article, the authors also give a non trivial proof that the problem 4.1 (B) is also not decidable for Petri nets.

There are usually two main approaches to circumvent undecidability limitations for solving verification problems. The first one is to extend the expressiveness of the models, starting from finite automata where opacity is decidable, in such a way that the opacity problem remains decidable. A second approach is to consider approximation techniques to address two goals:

- provide sound arguments that opacity holds but possibly failing at exhibiting counterexamples when such arguments do not hold;
- from an attacker point of view, create sound monitors for detecting information flow and possibly loosing the possibility to claim that opacity holds when no information flow is detected.

In the thesis, we will not investigate classes of models, other that finite automata, where opacity is decidable. We propose instead a general framework to reason about opacity using approximation techniques. We will now try to motivate why an approach based on approximation is interesting in the context of security analysis.

In [BKMR08], the authors introduced the notion of *uo-opacity* (for under/over-opacity) to handle approximations. Given a secret predicate $\phi \in \Phi$, the notion of uo-opacity consists in considering underapproximations and overapproximations of $\mathcal{R}(M)$ and ϕ. Then, according to three relations relating $\mathcal{R}(M)$ and ϕ to their approximations, we can conclude the ϕ-opacity for p_a. The uo-opacity property then is a sufficient condition for opacity and the authors applied this approach for verifying uo-opacity on Petri nets using *coverability graphs*. But uo-opacity may fail to provide a counterexample for opacity when uo-opacity is not satisfied. Furthermore, there is no generic approach to prove the inclusion relations of the uo-opacity. Here, we will focus on the detection of counterexamples, in order to take into account the following remark.

Remark 4.3 *Given an observed trace $\nu \in \Lambda_a^*$, the secret ϕ is preserved for an attacker $\mathcal{A}$ whenever*

- *$\mathcal{A}$ can prove that $\nu \notin DTraces(\mathcal{R}(M), p_a)(\phi)$,*
- *or $\mathcal{A}$ cannot prove that $\nu \in DTraces(\mathcal{R}(M), p_a)(\phi)$.*

For example, the second case will be typically true whenever it cannot be decided whether "$\nu \in DTraces(\mathcal{R}(M), p_a)(\phi)$". But, also, an attacker reasoning by approximating the behavior of M may not be able to infer secret information when the approximations are not precise enough. In some cases, there might exist no approximation techniques with a reasonable complexity which can help to prove whether $\nu \in DTraces(\mathcal{R}(M), p_a)(\phi)$. Therefore, the system may be considered as secure regarding the class of attackers related to the approximation techniques. When either one of the two situations of Remark 4.3 is true for every observation $\nu \in p_a(\mathcal{R}(M))$, then the secret predicate ϕ is never disclosed. Note that this is an important difference with safety conditions for example since a safety property can be violated disregarding what an external observer can observe or infer.

We borrow some ideas from uo-opacity but we follow a different presentation in order to connect the previous techniques for constructing monitors with an abstract interpretation framework based on Galois connections [CC77a, CC92a, Mas08]. Following [GM04], we place approximations as a part of the attacker's model similarly to what is done when considering models of attackers with limited computational resources for the verification of cryptographic protocols. In this context, the decision problem $\nu \in DTraces(\mathcal{R}(M), p_a)(\phi)$ may be decidable but can be very expensive. If this holds for every observation, the system can then be granted a certain level of confidentiality. We will not consider this aspect in the thesis. The work presented below has been published in [Dub09].

4.3.1 Basics of Abstract Interpretation

Verification problems often involve least or greatest fixpoint computation, typically to obtain the iterations of the operator *post*[1] and check whether some configurations are reachable. According to Theorem 2.2 (Knaster-Tarski), such fixpoint exists as soon the iterated operator is monotone within a complete lattice. But, this fixpoint computation may not terminate after a finite number of iterations. For example, we will see that

[1]more precisely $post(\Lambda)$ with our definition of *post*

the developments of the Chapter 5, about controller synthesis for opacity, address a non-termination problem of a fixpoint computation over the lattice of the sublanguages of $\mathcal{L}(M)$. We have seen in Chapter 2 that the computation of the least/greatest fixpoint always terminates if the underlying lattice is of finite height for example. So the idea of abstract interpretation is to approximate the fixpoint computation of an operator over a complete lattice by computing the exact fixpoint of an abstract operator defined over a lattice where fixpoint computations always terminate.

In this section, the objective is not to apply the general abstract interpretation theory to the case of opacity properties but to investigate the specificities of opacity regarding approximation techniques considering, as an example, a simple framework of abstract interpretation based on Galois connections. The development of opacity analysis techniques to more precise abstract interpretation techniques like the widening/narrowing approaches [CC77a, CC92a, CC92b] is not considered here.

We provide now the basic aspect of Galois connection based abstract interpretation. One can refer to [Mas05] for a more complete presentation. The original lattice, where the fixpoint computation problem arises, is denoted $(C, \sqsubseteq, \sqcap, \sqcup, \bot, \top)$ and called the concrete lattice (typically $\mathcal{P}(S)$). The abstract lattice, denoted $(A, \sqsubseteq^\sharp, \sqcap^\sharp, \sqcup^\sharp, \bot^\sharp, \top^\sharp)$ represents an abstract representation of the elements of C and we consider a Galois connection

$$(C, \sqsubseteq) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (A, \sqsubseteq^\sharp)$$

relating C and A with the semantics: for $x \in C$, $\alpha(x)$, called the abstraction of x, is the most precise approximation of x in A; for $y \in A$, $\gamma(y)$, called the concretization of y, is the greatest element of C that is approximated by y in C. Naturally, there is a loss of precision by manipulating the values of C in A, in the sense that for every $x \in C$, $x \sqsubseteq \gamma \circ \alpha(x)$. Also, for $y \in A$, $\alpha \circ \gamma(y) \sqsubseteq y$. In other words, $\gamma \circ \alpha$ is *extensive* whereas $\alpha \circ \gamma$ is *reductive.* Note that since (α, γ) defines a Galois connection, $\alpha(\bot) = \bot^\sharp$.

Example 4.2 *Suppose that we want to approximate the values of variables over the rational numbers. Therefore the concrete lattice is $\mathcal{P}(\mathbb{Q})$. Consider the abstract lattice given by the finite set*

$$A = \{\emptyset\} \cup \{[a, b] : a, b \in \{-\infty, -10, -9, \dots, 0, \dots, 9, 10, +\infty\}, a \leq b, a < +\infty, b > -\infty\}$$

We can define a Galois connection by $\gamma([a, b]) = [a, b]$ and:

$$\begin{array}{rccl} \alpha : & \mathcal{P}(\mathbb{Q}) & \rightarrow & A \\ & \emptyset & \mapsto & \emptyset \\ & X & \mapsto & [a, b] \textit{ where} \end{array}$$

where $a = -\infty$ if $min(X) < -10$, 10 if $min(X) > 10$ or $min(X)$ otherwise; and $b = -10$ if $max(X) < -10$, $+\infty$ if $max(X) > 10$ or $max(X)$ otherwise.

Consider now a monotone operator $op : C \to C$ and the problem of computing $lfp(op)$. The following will also apply for the computation of $gfp(op)$. In practice, the problem is often to compute $\sqcup\{f^i(x) : i \in \mathbb{N}\}$, where $x \in C$, and the operator op is then defined by $op(z) = x \sqcup f(z)$ following Proposition 2.2. The main idea of abstract interpretation is to compute $lfp(op^\sharp)$ approximating $lfp(op)$ by defining A such that the computation of lfp is always possible. To this effect, we will suppose that the abstract lattice A is of finite height.

Definition 4.2 (Sound approximation) *Given a monotone operator $op : C \to C$, a sound approximation of op in A is an operator $op^\sharp : A \to A$ such that for all $x \in C$,*

$$op(x) \sqsubseteq \gamma \circ op^\sharp \circ \alpha(x)$$

The best sound approximation of op in A is $op^\sharp_{best} = \alpha \circ op \circ \gamma$. But, for complexity reasons, it may sometimes be interesting to consider less precise abstractions of op.

Proposition 4.6 *If A is of finite height, then $lfp(op^\sharp)$ is effectively computable and is a sound approximation of $lfp(op)$, i.e. $lfp(op) \sqsubseteq \gamma(lfp(op^\sharp))$.*

Proof. First, we prove by induction that for all $n \in \mathbb{N}$, $op^n(\bot) \sqsubseteq \gamma \circ {op^\sharp}^n(\bot^\sharp)$. This is true for $n = 0$, according to Definition 4.2. Suppose that this property holds for $n \in \mathbb{N}$.

$$\begin{aligned} op^{n+1}(\bot) &= op(op^n(\bot)) \\ &\sqsubseteq op(\gamma \circ {op^\sharp}^n \circ \alpha(\bot)) \text{ since } op \text{ is monotone} \\ &\sqsubseteq \gamma \circ op^\sharp \circ \alpha \circ \gamma \circ {op^\sharp}^n(\bot^\sharp) \text{ by definition 4.2} \\ &\sqsubseteq \gamma \circ {op^\sharp}^{n+1}(\bot^\sharp) \text{ since } \alpha \circ \gamma \text{ is reductive} \end{aligned}$$

Then, for all $n \in \mathbb{N}$, $op^n(\bot) \sqsubseteq \gamma \circ {op^\sharp}^n(\bot^\sharp)$. Applying Theorem 2.2, for all $n \in \mathbb{N}$, $op^n(\bot) \sqsubseteq \gamma(lfp(op^\sharp))$ and then, again by Theorem 2.2, $lfp(op) \sqsubseteq \gamma(lfp(op^\sharp))$. $\square$

Remark 4.4 *A similar framework can be applied for underapproximation: the concrete operator is defined over A where the fixpoint computations may be impossible and the lattice C, for example of finite height, can be used to underapproximate fixpoints defined in A.*

Verifying Safety Properties with Abstract Interpretation We will make a comparison between opacity and safety properties, in order to outline the specificities of opacity regarding the application of approximation techniques.

A safety property expresses that nothing goes wrong in a system [Lam77]. For example, there is no division by zero during a program execution or, in a security context, malicious users cannot remove protected files.

A safety property depends on a predicate ψ defined over $E(\Lambda, S)$ such that if $r \in E(\Lambda, S)$, with $r \models \neg\psi$ then for all $r' \in E(\Lambda, S)$ with $r \leq r'$ then $r' \models \neg\psi$. This expresses the idea that if something bad happens on an execution of M, then it remains bad on every subsequent executions. A system satisfies a safety property if every execution satisfies the safe predicate:

Definition 4.3 (Safety) *We say that a system M is ψ-safe when for all $r \in \mathcal{R}(M)$, $r \models \psi$.*

The construction of Proposition 3.7 can also be applied for trace-based safety predicate so we consider in this chapter that the predicate ψ is state-based, i.e. there exists $F(\psi) \subseteq S$ s.t. $r \models \psi$ if $lst(r) \in F(\psi)$. The problem is then:

Problem 4.2 (Safety verification problem)

- ***Input:*** *An LTS $M = (\Lambda, S, \delta, S_0)$ and a state-based safe predicate ψ.*
- ***Problem:*** *Is M ψ-safe ?*

Hence, we have to check whether for every $r \in \mathcal{R}(M)$, $lst(r) \in F(\psi)$ which is done by verifying whether $(\cup\{post(\Lambda)^i(S_0) : i \in \mathbb{N}\}) \cap F(\psi) = \emptyset$. As $post(\Lambda)$ is ω-continuous, we can apply Proposition 2.2 and M is ψ-safe if and only if $reach(\Lambda)(S_0) \cap F(\psi)$ is empty[2]. As $reach(\Lambda)(S_0)$ may not be computable in $\mathcal{P}(S)$, we consider $(Q^\sharp, \sqsubseteq^\sharp, \sqcap^\sharp, \sqcup^\sharp, \top^\sharp, \bot^\sharp)$ to be a complete lattice of finite height whose elements are overapproximations of the sets of states of S via a Galois connection $(\mathcal{P}(S), \subseteq) \underset{\alpha^\sharp}{\overset{\gamma^\sharp}{\leftrightarrows}} (Q^\sharp, \sqsubseteq^\sharp)$. Then, $\alpha^\sharp(\emptyset) = \bot^\sharp$. Define now the operator $op : Y \mapsto S_0 \cup post(\Lambda)(Y)$ and let $op^\sharp : Q^\sharp \to Q^\sharp$ be a sound approximation of op. We can effectively compute $lfp(op^\sharp)$ which implies the following proposition and by consequence a practical technique to verify safety.

Proposition 4.7 *$\gamma(lfp(op^\sharp)) \cap F(\psi) = \emptyset$ implies that M is ψ-safe.*

Proof. Follows directly that $lfp(op) \subseteq \gamma(lfp(op^\sharp))$. □

In this section, we will investigate the computation of the monitors suggested at the end of the previous section 4.1 using the abstract interpretation techniques described above. To compute such a monitor, the objective is to obtain an overapproximation of the map $\Delta_a : \Lambda_a^* \to \mathcal{P}(S)$. As opacity depends on the attacker's partial observation, which is event based, we have to consider the iteration of $post$ with respect to a particular subset of events. For this, we will need to extend the framework presented above.

[2]Recall that $reach(\Lambda)(S_0) = lfp(Y \mapsto S_0 \cup post(\Lambda)(Y))$.

More on Galois Connection Based Abstract Interpretation Let $(C, \sqsubseteq) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (A, \sqsubseteq^\sharp)$ be an abstract interpretation framework as presented above. Given a monotone and ω-continuous map $f : C \to C$, we will see how to compute the closure operator $f^\uparrow : C \to C$, $x \mapsto lfp(z \mapsto x \sqcup f(z))$ as defined in Chapter 2[3]. This is provided by the following proposition.

Proposition 4.8 *If $f^\sharp$ is a sound approximation of f then $f^{\sharp\uparrow}$ is a sound approximation of $f^\uparrow$.*

Proof. Let $x \in C$ and define the map $g_x : C \to C$, $z \mapsto x \sqcup f(z)$. In that case, $f^\uparrow(x) = lfp(g_x)$. Define for all $y \in A$, $g^\sharp_y : A \to A$, $z \mapsto y \sqcup^\sharp f^\sharp(z)$. As $\gamma \circ \alpha$ is extensive, $x \sqsubseteq \gamma \circ \alpha(x)$. Also, for $z \in C$, $f(z) \sqsubseteq \gamma \circ f^\sharp \circ \alpha(z)$, since $f^\sharp$ is a sound approximation of f. Then $x \sqsubseteq \gamma \circ \alpha(x) \sqcup \gamma \circ f^\sharp \circ \alpha(z)$ and $f(z) \sqsubseteq \gamma \circ \alpha(x) \sqcup \gamma \circ f^\sharp \circ \alpha(z)$ so $x \sqcup f(z) \sqsubseteq \gamma \circ \alpha(x) \sqcup \gamma \circ f^\sharp \circ \alpha(z)$. Then, $x \sqcup f(z) \sqsubseteq \gamma(\alpha(x) \sqcup^\sharp f^\sharp(\alpha(z)))$, i.e. $g_x(z) \sqsubseteq \gamma \circ g^\sharp_{\alpha(x)} \circ \alpha(z)$ and $g^\sharp_{\alpha(x)}$ is a sound approximation of g_x. So $f^\uparrow(x) \sqsubseteq \gamma \circ f^{\sharp\uparrow} \circ \alpha(x)$. This holds for every $x \in C$ so $f^{\sharp\uparrow}$ is a sound approximation of $f^\uparrow$. $\square$

We will see now how this can be applied to the construction of sound monitors for opacity.

4.3.2 Construction of Monitors for Opacity

First, we remark that according to the definition of *reach*: for every $B \subseteq \Lambda$, $reach(B) = post(B)^\uparrow$. So approximating $post(B)$ will directly provide an approximation of $reach(B)$ by applying Proposition 4.8.

As we have done for the verification of safety properties, we consider an abstract lattice $(Q^\sharp, \sqsubseteq^\sharp, \sqcap^\sharp, \sqcup^\sharp, \top^\sharp, \bot^\sharp)$ representing sets of states and a Galois connection

$$(\mathcal{P}(S), \subseteq) \underset{\alpha^\sharp}{\overset{\gamma^\sharp}{\leftrightarrows}} (Q^\sharp, \sqsubseteq^\sharp)$$

For simplification, we consider here the best approximation of $post(B)$ defined with respect to this Galois connection but the results presented above can be adapted to other approximations. Formally, let

$$\begin{array}{rccl} post^\sharp : & \mathcal{P}(\Lambda) & \to & (Q^\sharp \to Q^\sharp) \\ & B & \mapsto & \alpha^\sharp \circ post(B) \circ \gamma^\sharp \end{array}$$

This implies that for all $B \subseteq \Lambda$ and $X \subseteq S$:

$$post(B)(X) \subseteq \gamma^\sharp \circ post^\sharp(B) \circ \alpha^\sharp(X)$$

[3]The approach is similar for $f^\downarrow$

It means that the set of states that are reachable according to *post* are also reachable according to the concretization of $post^\sharp$ after abstracting the parameters. We suppose now that for every $B \subseteq \Lambda$, $post^\sharp(B)$ is monotone and ω-continuous and we define the operator:

$$\begin{array}{rlcl} reach^\sharp: & \mathcal{P}(\Lambda) & \rightarrow & (Q^\sharp \rightarrow Q^\sharp) \\ & B & \mapsto & post^\sharp(B)^\uparrow \end{array}$$

As Λ may be infinite, and then M infinitely branching, it might also be necessary to approximate the set of labels, which can represent for example time information, communications with a parameter ranging over an infinite domain like `receive(m, A)` and `send(m, A)` of Example 1.1 or floating point values read from sensors. For this purpose, let θ be an equivalence relation over Λ such that the quotient set $\Sigma^\sharp \overset{def}{=} \theta(\Lambda)$ is a finite set of abstractions (equivalence classes) of the events of M. We suppose that θ refines the partition induced by the partial observability: $\forall\lambda \in \Lambda$, $\theta(\lambda) \subseteq \Lambda_a$ or $\theta(\lambda) \subseteq \Lambda_{ua}$. We note then $\Sigma^\sharp_a = \theta(\Lambda_a)$ and $\Sigma^\sharp_{ua} = \theta(\Lambda_{ua})$. The map θ is extended to words over Λ by $\theta(\epsilon) = \epsilon$, and for $w \in \Lambda^*$, $\lambda \in \Lambda$, $\theta(w\lambda) = \theta(w)\theta(\lambda)$.

Example 4.3 *To apply this in the case of Example 1.1, it would be relevant to define θ such that $\theta($`send(0, A)`$) = \{$`send(0, A)`$\}$ and if $m > 0$, $\theta($`send(m, A)`$) = \{$`send(k, A)`$: k > 0\}$.*

Remark 4.5 *Note that the approximation of the set of events can equivalently be defined in terms of equivalence relation or in terms of Galois connection. Indeed, a Galois connection based approach can be defined equivalently in terms of closure operator. And, given a set A, there exists a lattice isomorphism between $Eq(A)$ and the set of upper closure operators over $\mathcal{P}(A)$ [Mas05].*

Proposition 4.9 *For all $B \subseteq \Lambda$ and all $X \subseteq S$,*

$$reach(B)(X) \subseteq \gamma^\sharp \circ reach^\sharp(\cup\theta(B)) \circ \alpha^\sharp(X)$$

Proof. As $reach^\sharp(B)$ is a sound approximation of $reach(B)$, $reach(B)(X) \subseteq \gamma^\sharp \circ reach^\sharp(B) \circ \alpha^\sharp(X)$. Also $B \subseteq \cup\theta(B)$ and $reach^\sharp$ is monotonic since it is the least fixpoint of a monotonic operator. The result follows. □

With these definitions, we can now compute an approximation of $Det_a(M)$. Similarly to $reach_{ua}$, we define $reach^\sharp_{ua} = reach^\sharp(\Sigma^\sharp_{ua})$. We define then the finite LTS $det^\sharp_a(M) = (\Sigma^\sharp_a, Q^\sharp, \Delta^\sharp_a, q^\sharp_0)$ by $q^\sharp_0 = reach^\sharp_{ua} \circ \alpha^\sharp(S_0)$, and

$$\begin{array}{rlcl} \Delta^\sharp_a: & \Sigma^\sharp \times Q^\sharp & \rightarrow & Q^\sharp \\ & \sigma,\ q & \mapsto & q' \text{ if } q' = reach^\sharp_{ua} \circ post^\sharp(\sigma)(q) \neq \bot^\sharp \end{array}$$

Note that $q_0^\sharp$ is a sound approximation of X_0 since $X_0 = \mathit{reach}_{ua}(S_0) \subseteq \gamma^\sharp \circ \mathit{reach}_{ua}^\sharp \circ \alpha^\sharp(S_0) = \gamma^\sharp(q_0^\sharp)$, so $\alpha^\sharp(X_0) \sqsubseteq^\sharp q_0^\sharp$. With this definition, the automaton $\mathit{det}_a^\sharp(M)$ is an abstraction of the automaton $\mathit{det}_a(M)$ defined in Definition 4.1, as demonstrated by the following proposition.

Proposition 4.10 *For all $\nu \in \Lambda_a^*$ and $X \subseteq S$, $\Delta_a(\nu, X) \subseteq \gamma^\sharp \circ \Delta_a^\sharp(\theta(\nu), \alpha^\sharp(X))$.*

Proof. The operators *post*, reach_{ua}, $\mathit{post}^\sharp$ and $\mathit{reach}_{ua}^\sharp$ are monotone. Let $\lambda \in \Lambda_a$ and $X \subseteq S$.

$$\begin{aligned}
\Delta_a(\lambda, X) &= \mathit{reach}_{ua} \circ \mathit{post}(\lambda)(X) \\
&\subseteq \mathit{reach}_{ua} \circ \gamma^\sharp \circ \mathit{post}^\sharp(\theta(\lambda)) \circ \alpha^\sharp(X) \text{ because } \mathit{post}^\sharp \text{ is sound} \\
&\subseteq \gamma^\sharp \circ \mathit{reach}_{ua}^\sharp \circ \alpha^\sharp \circ \gamma^\sharp \circ \mathit{post}^\sharp(\theta(\lambda)) \circ \alpha^\sharp(X) \text{ (Prop. 4.9)} \\
&\subseteq \gamma^\sharp \circ \mathit{reach}_{ua}^\sharp \circ \mathit{post}^\sharp(\theta(\lambda)) \circ \alpha^\sharp(X) \\
&\subseteq \gamma^\sharp \circ \Delta_a^\sharp(\theta(\lambda), \alpha^\sharp(X))
\end{aligned}$$

Now, let $\nu = \lambda_1\lambda_2 \ \ldots \ \lambda_n \in \Lambda_a^*$.

$$\begin{aligned}
\Delta_a(\nu, X) &= \Delta_a(\lambda_n, \cdot) \circ \ \ldots \ \circ \Delta_a(\lambda_2, \cdot) \circ \Delta_a(\lambda_1, \cdot)(X) \\
&\subseteq \Delta_a(\lambda_n, \cdot) \circ \ \ldots \ \circ \Delta_a(\lambda_2, \cdot) \circ \gamma^\sharp \circ \Delta_a^\sharp(\theta(\lambda_1), \cdot) \circ \alpha^\sharp(X) \\
&\subseteq \gamma^\sharp \circ \Delta_a^\sharp(\theta(\lambda_n), \cdot) \circ \ \ldots \ \circ \alpha^\sharp \circ \gamma^\sharp \circ \Delta_a^\sharp(\theta(\lambda_1), \cdot) \circ \alpha^\sharp(X) \\
&\subseteq \gamma^\sharp \circ \Delta_a^\sharp(\theta(\lambda_n), \cdot) \circ \ \ldots \ \circ \Delta_a^\sharp(\theta(\lambda_2), \cdot) \circ \Delta_a^\sharp(\theta(\lambda_1), \cdot) \circ \alpha^\sharp(X) \\
&\subseteq \gamma^\sharp \circ \Delta_a^\sharp(\theta(\nu), \alpha^\sharp(X))
\end{aligned}$$

□

The following lemma will be used to simplify the proofs of Proposition 4.10 and Theorem 4.3.

Lemma 4.2 *For all $\nu \in \Lambda_a^*$, if $\nu \in p_a(\mathcal{R}(M))$, then $\Delta_a(\nu, X_0) \subseteq \gamma^\sharp \circ \Delta_a^\sharp(\theta(\nu), q_0^\sharp)$ where $X_0 = \mathit{reach}_{ua}(S_0)$.*

Proof. If $\nu \in p_a(\mathcal{R}(M))$, then $\Delta_a(\nu, X_0)$ is defined. Moreover, according to Proposition 4.10, $\Delta_a(\nu, X_0) \subseteq \gamma^\sharp \circ \Delta_a^\sharp(\theta(\nu), \alpha^\sharp(X_0))$. The operator $\Delta_a^\sharp(\theta(\nu), \cdot)$ is monotone because $\mathit{post}^\sharp$ and $\mathit{reach}_{ua}^\sharp$ are monotone. Then, $\alpha^\sharp(X_0) \sqsubseteq^\sharp q_0^\sharp$ implies that $\Delta_a^\sharp(\theta(\nu), \alpha^\sharp(X_0)) \sqsubseteq^\sharp \Delta_a^\sharp(\theta(\nu), q_0^\sharp)$. Finally, $\Delta_a(\nu, X_0) \subseteq \gamma^\sharp \circ \Delta_a^\sharp(\theta(\nu), q_0^\sharp)$. □

The following proposition details the way this approximation framework provides an abstraction of the set of observed traces.

Corollary 4.4 *For all* $\nu \in \mathcal{L}(det_a(M))$, $\theta(\nu) \in \mathcal{L}(det_a^\sharp(M))$.

Proof. Following the definition of Δ_a, $\nu \in \mathcal{L}(det_a(M))$ if and only if $\Delta_a(\nu, X_0) \neq \emptyset$. According to Lemma 4.2, $\Delta_a^\sharp(\theta(\nu), q_0^\sharp) \neq \bot^\sharp$ so $\theta(\nu) \in \mathcal{L}(det_a^\sharp(M))$. □

We now define the set of accepting states of $det_a^\sharp(M)$ given by the map:

$$\begin{array}{rrcl} F^\sharp : & \Phi & \rightarrow & Q^\sharp \\ & \phi & \mapsto & \{q \in Q^\sharp : \gamma^\sharp(q) \subseteq F(\phi)\} \end{array}$$

We relate the words accepted by the automaton to the observed traces such that secret information is disclosed:

Theorem 4.3 *Given* $\phi \in \Phi$, *for all* $\nu \in p_a(\mathcal{R}(M))$ *such that* $\theta(\nu) \in \mathcal{L}(det_a^\sharp(M), q_0^\sharp, F^\sharp(\phi))$, *we can deduce that* $\nu \in DTraces(\mathcal{R}(M), p_a)(\phi)$.

Proof. Let $\phi \in \Phi$. Let $r \in \mathcal{R}(M)$ such that $p_a(r) = \nu$. According to Corollary 4.2, this implies that $\Delta_a(\nu, X_0) \neq \emptyset$. Suppose that $\theta(\nu) \in \mathcal{L}(det_a^\sharp(M), q_0^\sharp, F^\sharp(\phi))$. In that case, $\Delta_a^\sharp(\theta(\nu), q_0^\sharp) \in F^\sharp(\phi)$, which means that $\gamma^\sharp(\Delta_a^\sharp(\theta(\nu), q_0^\sharp)) \subseteq F(\phi)$. According to Lemma 4.2, we obtain that $\Delta_a(\nu, X_0) \subseteq F(\phi)$. Finally, according to Proposition 4.2, $\nu \in DTraces(\mathcal{R}(M), p_a)(\phi)$. □

This result provides an effective methodology to monitor information flow as soon as the Galois connection is given. Practically, the attacker observes a trace ν from M, computes the word $\theta(\nu)$ and when $\theta(\nu)$ is accepted by $det_a^\sharp(M)$, the attacker knows that ϕ is true on the current run executed in M. This can be formalized by the following monitor:

$$\begin{array}{rrcl} \Gamma_\Phi^\sharp : & \Lambda_a^* & \rightarrow & \mathcal{P}(\Phi) \\ & \nu & \mapsto & \{\phi \in \Phi : \Delta_a^\sharp(\theta(\nu), q_0^\sharp) \in F^\sharp(\phi)\} \end{array}$$

This monitor is sound according to Theorem 4.3. But of course, this monitor cannot recognize all the traces of $DTraces(\mathcal{R}(M), p_a)(\Phi)$, i.e. there may exist ν in this set such that $\Delta_a^\sharp(\theta(\nu), q_0^\sharp) \notin F^\sharp(\phi)$.

Given a secret $\phi \in \Phi$, the set of observed traces such that this secret is disclosed contains the set of traces $\nu \in p_a(\mathcal{R}(M))$ such that $\theta(\nu)$ reaches a state of $F^\sharp(\phi)$ in $\mathrm{det}_a^\sharp(M)$. But, a word $\mu \in \mathcal{L}(det_a^\sharp(M), q_0^\sharp, F^\sharp(\phi))$ cannot be exhibited as a proof of the non-opacity of ϕ. The reason is that it cannot always be decided a priori that an element of the class of events represented by μ (i.e. a trace ν such that $\theta(\nu) = \mu$) exists in $\mathcal{L}(det_a(M))$. Also, on opposition to the analysis of safety properties, we cannot conclude from $\mathcal{L}(det_a^\sharp, p_a, \phi) = \emptyset$ that M is ϕ-opaque. In particular, it may happen that reasoning with a more precise abstract interpretation framework allows an attacker to infer secret information.

In the next section, we will see how considering underapproximation can help to statically prove that a system is not opaque.

4.3.3 Static Computation of Vulnerabilities Combining Under and Over Approximations

In this section, we will see how underapproximating the set of observed traces of M can help to statically exhibit observed traces disclosing secret information. For this purpose, let $(Q^\flat, \sqsubseteq^\flat, \sqcap^\flat, \sqcup^\flat, \top^\flat, \bot^\flat)$ be a finite lattice representing an underapproximation of the sets of states and let $(Q^\flat, \sqsubseteq^\flat) \underset{\gamma^\flat}{\overset{\alpha^\flat}{\longleftrightarrow}} (\mathcal{P}(S), \subseteq)$ be a Galois connection such that $\alpha^\flat \circ \gamma^\flat = id_{Q^\flat}$. To underapproximate the observable events, we consider a finite subset $\Sigma_a^\flat \subseteq \Lambda_a$. Finally, we consider the sound underapproximation $post^\flat$ of the operator $post$ defined by:

$$\begin{array}{rccl} post^\flat : & \mathcal{P}(\Lambda) & \rightarrow & (Q^\flat \rightarrow Q^\flat) \\ & B & \mapsto & \alpha^\flat \circ post(B) \circ \gamma^\flat \end{array}$$

and define

$$\begin{array}{rccl} reach^\flat : & \mathcal{P}(\Lambda) & \rightarrow & (Q^\flat \rightarrow Q^\flat) \\ & B & \mapsto & post^\flat(B)^\uparrow \end{array}$$

We also denote by $reach^\flat_{ua}$ the operator $reach^\flat(\Sigma^\flat_{ua})$. Similarly to $det^\sharp_a(M)$, we define the finite LTS $det^\flat_a = (\Sigma^\flat, Q^\flat, \Delta^\flat_a, q^\flat_0)$ where $q^\flat_0 = reach^\flat_{ua} \circ \alpha^\flat(S_0)$ (note that $q^\flat_0 \sqsubseteq^\flat \alpha^\flat(X_0)$ this time) and

$$\begin{array}{rccl} \Delta^\flat_a : & \Sigma^\flat \times Q^\flat & \rightarrow & Q^\flat \\ & \lambda, q & \mapsto & reach^\flat_{ua} \circ post^\flat(\lambda)(q) \text{ if not } \bot^\flat \end{array}$$

We show now how to relate this definition of $det^\flat_a(M)$ with $\mathcal{L}(det_a(M))$.

Proposition 4.11 *For all $\nu \in \Sigma^{\flat^*}$ and $X \subseteq S$, $\gamma^\flat \circ \Delta^\flat_a(\nu, \alpha^\flat(X)) \subseteq \Delta_a(\nu, X)$.*

Proof. The proof is similar to the proof of Proposition 4.10. □

Let $F^\flat = \{q \in Q^\flat : \gamma^\flat(q) \neq \emptyset\}$, the set of states of $Q^\flat$ such that their concretization is not empty.

Corollary 4.5 *$\mathcal{L}(det^\flat_a(M), q^\flat_0, F^\flat) \subseteq \mathcal{L}(det_a(M))$.*

Proof. Let $\nu \in \mathcal{L}(det^\flat_a(M), q^\flat_0, F^\flat)$. Then $\gamma^\flat \circ \Delta^\flat_a(\nu, q^\flat_0) \neq \emptyset$. Since $q^\flat_0 \sqsubseteq^\flat \alpha^\flat(X_0)$, $\gamma^\flat \circ \Delta^\flat_a(\nu, \alpha^\flat(X_0)) \neq \emptyset$ by monotony. So, according to Proposition 4.11, $\Delta_a(\nu, X_0) \neq \emptyset$. We conclude then that $\nu \in \mathcal{L}(det_a(M))$. □

Proposition 4.12 *Let $\phi \in \Phi$. For all $\nu \in \mathcal{L}(det_a^\flat(M), q_0^\flat, F^\flat)$, if $\theta(\nu) \in \mathcal{L}(det_a^\sharp(M), q_0^\sharp, F^\sharp(\phi))$, then $\nu \in DTraces(\mathcal{R}(M), \pi_a)(\phi)$.*

Proof. Follows Corollary 4.5 and Theorem 4.3. □

Now, in the context where both $(Q^\flat, \sqsubseteq^\flat)$ and $(Q^\sharp, \sqsubseteq^\sharp)$ are finite lattices, we can construct the finite deterministic LTS $G^\flat = (\Sigma^\flat, Q^\flat \times Q^\sharp, \delta^\flat, (q_0^\flat, q_0^\sharp))$ where $\delta^\flat$ is defined by $\delta^\flat(q^\flat, q^\sharp) = (\Delta_a^\flat(\lambda, q^\flat), \Delta_a^\sharp(\theta(\lambda), q^\sharp))$. Then, the problem of computing attack scenarios, i.e. observed traces such that the secret is disclosed, is reduced to a reachability problem. For this let $Q_f^{G^\flat} = \{(q^\flat, q^\sharp) \in Q^\flat \times Q^\sharp : q^\flat \in F^\flat \ \wedge \ q^\sharp \in F^\sharp(\phi)\}$

Theorem 4.4 *$\mathcal{L}(G^\flat, (q_0^\flat, q_0^\sharp), Q_f^{G^\flat}) \subseteq DTraces(\mathcal{R}(M), p_a)(\phi)$.*

Proof. Follows by applying Proposition 4.12. □

This theorem provides effective methods to statically generate information flow attacks using abstract interpretation. In other words, it allows to provide negative answer to the Problem 4.1 (B). A possible application of this result would be to automate the computation of vulnerabilities on critical software.

4.4 Language Based Approach and Regular Abstractions

In this section, we consider a system M where the set of events is a finite alphabet Σ. We do not assume that the set of states is finite, so the language $\mathcal{L}(M)$ is not necessarily regular. We consider in this section that the secret information is given by a finite set of trace-based secret predicates Φ. As in the previous section, the attacker observes the events $\Sigma_a \subseteq \Sigma$ and we denote the projection by $\pi_a = \pi_{\Sigma \to \Sigma_a}$. In this section, we will consider provided a *regular abstraction* of M, i.e. a finite LTS G such that $\mathcal{L}(M) \subseteq \mathcal{L}(G)$ and we address two problems: first, we will see how an attacker can construct sound monitors based on G; second, based on this abstraction G, we will see how a supervisor can detect the occurrences of information flow using the *diagnosis theory.*

As the secret predicates are trace-based, we can follow a language based approach, thanks to the following remark.

Remark 4.6 *Given two sets of states S and S' and two systems, represented by their semantics, $R \subseteq E(\Lambda, S)$ and $R' \subseteq E(\Lambda, S')$ generating the same language, i.e. $tr(R) = tr(R') \subseteq \Sigma^*$. Then for all $r \in R$ and all $r' \in R'$,*

$$tr(r) = tr(r') \implies (r \in Disclose(R, p_a)(\phi) \iff r \in Disclose(R', p'_a)(\phi))$$

where $p_a : E(\Lambda, S) \to \Sigma_a^*,\ r \mapsto \pi_a \circ tr(r)$ and $p'_a : E(\Lambda, S') \to \Sigma_a^*$ is defined similarly.

This also implies that

$$DTraces(R, p_a)(\phi) = DTraces(R', p_a)(\phi)$$

Therefore, in the case of trace-based secret predicates, we can forget the states of M and consider only its generated language. We will denote then $Disclose(L, \pi_a)(\phi) \subseteq \Sigma^*$ the set of counterexamples and $DTraces(L, \pi_a)(\phi)$ instead of $Disclose(R, p_a)(\phi)$ and $DTraces(R, p_a)(\phi)$ when $L = tr(R)$.

We now give some useful results of language based opacity and apply them to investigate the opacity problem with regular abstractions. We start by giving a language characterization of the set of counterexamples.

Proposition 4.13 *Let $L \subseteq \Sigma^*$ be a prefix-closed language.*

$$Disclose(L, \pi_a)(\phi) = L \setminus \pi_a^{-1}(\pi_a(L \setminus L(\phi)))$$

Proof. $\pi_a^{-1}(\pi_a(L \setminus L(\phi)))$ is the set of words of L that are observationally equivalent to a word of L that is not in $L(\phi)$. Then, when a word of L is not in this set, this word discloses the secret. □

Also, $Disclose(L, \pi_a)(\Phi) = L \setminus \cup\{\pi_a^{-1}(\pi_a(L \setminus L(\phi))) : \phi \in \Phi\}$. By consequence, when L is regular, $Disclose(L, \pi_a)(\Phi)$ and $L \setminus Disclose(L, \pi_a)(\Phi)$ are also regular.

The following proposition will be important for this section and also for the next chapter. It shows the effect of language inclusion on opacity. The following proposition is a reformulation in terms of languages of Proposition 3.4.

Proposition 4.14 *If L_1 and L_2 are two languages such that $L_1 \subseteq L_2$, then*

$$L_1 \cap Disclose(L_2, \pi_a)(\phi) \subseteq Disclose(L_1, \pi_a)(\phi)$$

This proposition can also be reformulated in terms of observed traces.

Corollary 4.6 *If $L_1 \subseteq L_2$,*

$$\pi_a(L_1) \cap DTraces(L_2, \pi_a)(\phi) \subseteq DTraces(L_1, \pi_a)(\phi)$$

But we cannot say more about opacity and inclusion as the following remark points out.

Remark 4.7 *In general, opacity is not preserved by inclusion. Consider the following examples where $\Sigma_a = \{a, b\}$, $\Sigma_{ua} = \{\tau, h\}$ and $L(\phi) = \Sigma^* h \Sigma^*$. Each time, $L_1 \subseteq L_2$ on the figures 4.2 and 4.3 displaying counterexamples.*

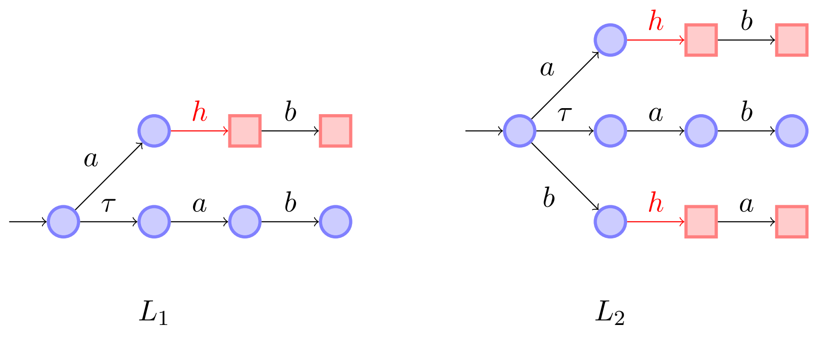


Figure 4.2: L_1 is opaque but not L_2

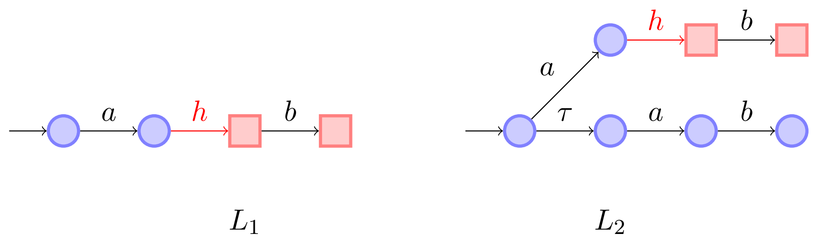


Figure 4.3: L_2 is opaque but not L_1

This also implies that opacity is generally not preserved by intersection.

4.4.1 Monitor for the Attackers

We investigate now how to construct sound monitors for the attacker using regular abstractions.

Problem 4.3

- ***Input:*** *A system given as an LTS M over a finite alphabet of events Σ, a finite set of regular predicates Φ and a regular abstraction $G = (\Sigma, Q, \delta, q_0)$ of M, i.e. $\mathcal{L}(M) \subseteq \mathcal{L}(G)$.*
- ***Problem:*** *Compute a sound monitor based on G.*

Applying Proposition 3.7, we can assume without lost of generality that there exists a map $F : \Phi \to \mathcal{P}(Q)$ such that the predicates of Φ are also state-based over $E(\Sigma, Q)$. Then, we can apply the results of section 4.2 and compute the set $DTraces(\mathcal{L}(G), \pi_a)(\phi)$, for all $\phi \in \Phi$. Now, applying Corollary 4.6, the attacker can construct a sound monitor based on G, thanks to the fact that when the attacker observes a trace $\mu \in \pi_a(\mathcal{L}(M))$,

$$\mu \in DTraces(\mathcal{L}(G), \pi_a)(\phi) \implies \mu \in DTraces(\mathcal{L}(M), \pi_a)(\phi)$$

In such a case, the secret ϕ is disclosed. We retrieve a notion similar to the one presented section 4.3.

In other words, the monitor constructed via the determinization of G is sound. But naturally, and as it was the case in Section 4.3.2: some traces of $\mathcal{L}(M)$ revealing a secret may not be discovered by this monitor and moreover, since the LTS M can be an infinite state system, it may be impossible to decide a priori, i.e. without executing the system, whether a given trace $\mu \in DTraces(\mathcal{L}(G), \pi_a)(\phi)$ is a real observed trace of M. Then, for the opacity verification problem, it may happen that the disclosing traces of G are false alarms. As done at the end of Section 4.3, we can consider a regular underapproximation of $\mathcal{L}(M)$ and then exhibit real observed traces disclosing the secret by applying Corollary 4.6. But such an approach will not be complete either as we will not be able to decide anything for traces that are not in the underapproximations.

4.4.2 Diagnosing Information Flow

In this section, we present an alternative solution to the combination of under and over-approximations. These techniques, based on Diagnosis Theory, aim to certify on-line that no information flow has occurred. We consider a supervisor D, called the diagnoser, also partially observing the events of $\Sigma_o \subseteq \Sigma$ via the projection $\pi_o = \pi_{\Sigma \to \Sigma_o}$, and whose objective is to detect at runtime the existence of security breaches (see the architecture on Figure 4.4). As this supervisor is partially observing, we will see that the techniques devel-

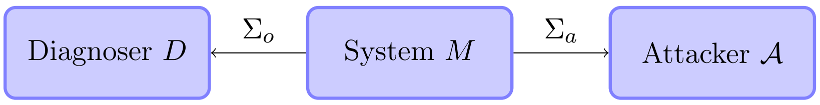


Figure 4.4: Diagnosing information flow

oped for monitoring opacity can also be applied to this end. We will see that if we accept that the occurrences of information can be detected after a delay, bounded and known a priori, we can apply diagnosis theory [SSL+95, SLS+96, JMPC06, JMGL08] and provide sufficient conditions about G such that every information flow is detected. The benefit of this approach is to invalidate the security of M at runtime and, if applied for a sufficiently long period, it will let aside, when they exist, the cases of information flow that are unlikely to happen with a standard use of M (i.e. with a probability that can be neglected) and then statistically granting the system a certain level of confidence in its security.

Basics of Diagnosis Theory The problem of diagnosis has been studied in the context of discrete event systems by [SSL+95, SLS+96, DLT00, QK04] and the objective is to detect failures in systems under partial observation. In these works, the failures are modeled by a subset of the events called the faulty events. The diagnosis theory has then be generalized in [KJ04, JMPC06]. In [KJ04], the authors extend diagnosis techniques to detect the validation of a subclass of LTL formulas. In [JMPC06], the authors develop diagnosis techniques to detect the violation of trace-based safety property. In this last paper, in order to model more realistic situations, the delays are defined over the length of observed traces instead of the length of words. We give next a brief overview of the diagnosis theory for safety properties before applying it to the detection of secret information flow.

The objective of diagnosis is to detect under partial observation the violation of a safety predicate. As the negations of safety predicates are extension-closed, i.e. every extension of a run violating a safety property also violates this safety property, the information “the property ψ has been violated” is equivalent to “the property ψ is violated”[4]. Then, detecting the violation of a safety property with a delay is acceptable as soon as this delay is known a priori. The fact that such delay should be known a priori expresses that when a safety property is possibly violated according to what has been observed, the diagnoser only needs to wait a known amount of time to be sure whether the safety property has been violated or not. We model the time passing by the length of the generated words and the delay is given as a bound on the word’s length.

Remark 4.8 *Note that our approach should gain in expressiveness with delays defined by counting the events of a subset $\Sigma_t \subseteq \Sigma_o$ where the events of Σ_t model for example clock ticks, thus generalizing the approach of [JMPC06]. This aspect is let for subsequent developments.*

The next definition formalizes the concept of diagnosability for safety properties.

Definition 4.4 (Diagnosability) *Given a trace-based safety predicate ψ, a prefix-closed language L is ψ-diagnosable if there exists $N \in \mathbb{N}$ such that for all $w \in L$,*

$$w \in L(\neg\psi) \implies (\forall w' \in w^{-1}L,\ |w'| \geq N \implies ww' \in \mathit{Disclose}(L, \pi_o)(\neg\psi))$$

We will say then that L is ψ-diagnosable with delay N when the delay is needed. Observe that if L is ψ-diagnosable with delay N and $N \leq N'$, then L is ψ-diagnosable with delay N'. As the objective is to apply diagnosis techniques to reason about opacity using regular abstractions, we investigate the effect of inclusion on diagnosability. The following proposition shows that diagnosability is preserved by inclusion.

Proposition 4.15 *Given a trace-based safety predicate ψ and two languages L_1 and L_2 such that $L_1 \subseteq L_2$. If L_2 is ψ-diagnosable then L_1 is ψ-diagnosable.*

[4]Note that this is not true for opacity as a secret predicate can be true for a particular run and false on its extensions.

Proof. Suppose that L_2 is ψ-diagnosable with delay N. Let $w \in L_1 \cap L(\neg\psi)$ and $w' \in w^{-1}L_1$ such that $|w'| \geq N$. Then $w \in L_2 \cap L(\neg\psi)$ and $w' \in w^{-1}L_2$. So $|w'| \geq N$ implies that $ww' \in Disclose(L_2, \pi_o)(\neg\psi)$. As $ww' \in L_1$, $ww' \in Disclose(L_1, \pi_o)(\neg\psi)$ according to Proposition 4.14. □

Remark 4.9 *Note that diagnosability is not preserved by union. Indeed, consider the alphabet $\Sigma = \{\tau, f, x\}$ with $\Sigma_o = \{x\}$ and the safe predicate ψ defined by $L(\psi) = \Sigma^* f \Sigma^*$. Then the two languages L_1 and L_2 depicted in figure 4.5 are both ψ-diagnosable, but the language $L_1 \cup L_2$ is not.*

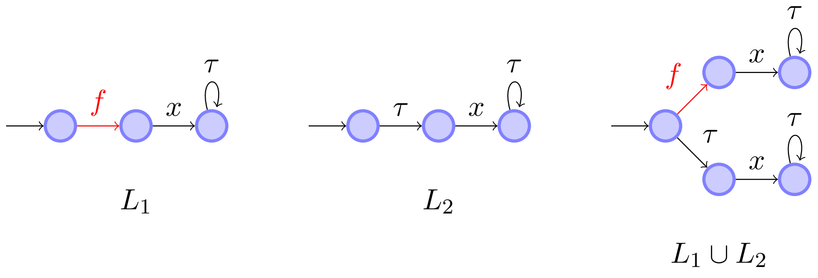


Figure 4.5: Union does not preserve diagnosability

We will now investigate the diagnosability verification problem for regular and prefix-closed languages and regular safety predicates. We can state the problem as follows.

Problem 4.4

- ***Input:*** *A prefix closed and regular language L and a regular safety predicate ψ.*
- ***Input:*** *Is L ψ-diagnosable ?*

To solve this problem, let $H = L(\neg\psi) \cap (L \setminus Disclose(L, \pi_o)(\neg\psi))$. The language H is the set of words (of Σ^*) such that ψ is violated but there exists at least one word of L with the same observation and not violating ψ, thus preventing a diagnoser observing through π_o to infer that ψ is violated. We will show that L is ψ-diagnosable if and only if there is no cycle within words of H. This language is regular as L and $L(\psi)$ are regular. Then, let $A = (\Sigma, Q, \delta, q_0, Q_f)$ be the minimal DFA accepting the language H[5].

Lemma 4.3 *L is not ψ-diagnosable if and only if*

$$\forall n \in \mathbb{N},\ \exists w \in H,\ \exists w' \in w^{-1}H,\ |w'| \geq n \tag{4.1}$$

[5]In that case, $Q = \{w^{-1}H : w \in \Sigma^*\}$ according to the classical construction of the minimal DFA based on the Myhill-Nerode equivalence relation [Ner58].

Proof. Suppose that L is not ψ-diagnosable. Writing the negation of Definition 4.4, we obtain:

$$\forall n \in \mathbb{N},\ \exists w \in L \cap L(\neg\psi),\ \exists w' \in w^{-1}L,\ |w'| \geq N \wedge ww' \notin \mathit{Disclose}(L, \pi_o)(\neg\psi))$$

Now note that if we had $w \in \mathit{Disclose}(L, \pi_o)(\neg\psi))$, then we would also have $ww' \in \mathit{Disclose}(L, \pi_o)(\neg\psi))$ as $L(\psi)$ is prefix closed. So $w \in (L \cap L(\neg\psi)) \setminus \mathit{Disclose}(L, \pi_o)(\neg\psi)) \subseteq H$. Also, as $w \in L(\neg\psi)$, we also have $ww' \in L(\neg\psi)$. Then $w \in H$ and $ww' \in H$ and the expression (4.1) follows. □

Note that this lemma is always true, even if L and $L(\psi)$ are not regular. The following lemma states that when the bound depends on the words violating ψ, then in the case of L and $L(\psi)$ regular, we can compute a global bound, that will apply for every faulty words, and therefore imply the ψ-diagnosability of L.

Lemma 4.4 *The language L is ψ-diagnosable if and only if*

$$\forall w \in H,\ \exists N_w \in \mathbb{N},\ \forall w' \in w^{-1}H,\ |w'| < N_w \tag{4.2}$$

Proof. It is clear that (4.2) holds when L is ψ-diagnosable, by rewriting the negation of (4.1) and inverting the first two quantifiers.

For the other implication, as L is regular, the language H is also regular so the minimal DFA A as above accepting the language H exists and $Q = \{w^{-1}H : w \in \Sigma^*\}$ is finite. Now note that in (4.2), N_w only depends on $q = w^{-1}H$. Then, writing $N(q) = N_w$ for all $w \in H$, we can define $N = max\{N(q) : q \in Q\}$. Therefore, the language L is ψ-diagnosable with delay N. □

Definition 4.5 (Accepting cycle) *Given an automaton $A = (\Sigma, Q, \delta, q_0, Q_f)$, an* accepting cycle *is a run $r \in \mathcal{R}(A)$, $r = q_0 \xrightarrow{\sigma_1} q_1 \xrightarrow{\sigma_2} \cdots \xrightarrow{\sigma_n} q_n$ such that for some $0 \leq i < n$, $q_i = q_n$, and for all j, $i \leq j \leq n$, $q_j \in Q_f$.*

Deciding the existence of an accepting cycle in a finite automata can be done in polynomial time [CLRS01]. The two lemmas above can now be applied to the diagnosability verification problem via the following proposition.

Proposition 4.16 *Given a regular and prefix-closed language L, L is ψ-diagnosable if and only if there is no accepting cycle in A where A is the minimal DFA accepting the language $H = L(\neg\psi) \cap (L \setminus \mathit{Disclose}(L, \pi_o)(\neg\psi))$.*

Proof.

- Assume that there exists an accepting cycle r in A, with $r = q_0 \xrightarrow{w} q_i \xrightarrow{w'} q_n$ and $q_i = q_n$ $(i < n)$. Let $N \in \mathbb{N}$. Then, $w \in H$ and $ww' \in H$. As $q_i = q_n$, $w^{-1}H = ww'^{-1}H$ and then $w^{-1}H = (ww'^{n})^{-1}H$ for all n in N. As $|w'| \geq 1$, $|w'^{N}| \geq N$ and $w'^{N} \in w^{-1}H$. Applying Lemma 4.3, L is not ψ-diagnosable.

- Suppose now that L is not ψ-diagnosable. Applying Lemma 4.4, the negation of (4.2) is
$$\exists w \in H,\ \forall n \in \mathbb{N},\ \exists w' \in w^{-1}H,\ |w'| \geq n$$
which implies, choosing $n = |Q|$, that
$$\exists w \in H,\ \exists w' \in w^{-1}H,\ |w'| \geq |Q| \tag{4.3}$$
If we let $q = w^{-1}H$, then as $|w'| \geq |Q|$, the run starting from q and generating the word w' in A must contain twice the same state. In other words, there exists $q' \in Q$ such that $r = q_0 \xrightarrow{w} q \xrightarrow{w_1} q' \xrightarrow{w_2} q' \in \mathcal{R}(A)$ with $w_1w_2 \leq w'$ and $|w_2| \geq 1$. Also, $w \in H$ implies that $q \in Q_f$. Note that $L(\psi)$ being prefix-closed, $ww_1 \in Disclose(L, \pi_o)(\neg\psi)$ would implies $ww' \in Disclose(L, \pi_o)(\neg\psi)$ and then $ww' \notin H$. So $w_1 \in w^{-1}H$ and therefore $q' \in Q_f$. The same argument holds for all $u \leq w_2$, implying that $\delta(u, q') \in Q_f$. So r is an accepting cycle in A.

□

So the proposition 4.16 implies the existence of a procedure to check that a regular and prefix-closed language is diagnosable. Therefore, the problem 4.4 is decidable. We will see now how to apply this techniques to the detection of information flow vulnerabilities using regular abstractions.

Application to Information Flow We consider the situation depicted in figure 4.6. The attacker still observes the events of $\Sigma_a \subseteq \Sigma$ and searches to infer the truth of the trace-based secret predicates of Φ using monitors based on the abstraction G of M. The supervisor D observes the events of $\Sigma_o \subseteq \Sigma$ and wants to infer, also on the basis of the abstraction G, whether the attacker, reasoning using G, could disclose some secret of Φ. We will see next how to apply the notion of diagnosability in the context of information flow. The problem we consider is:

Problem 4.5

- ***Input:*** *A system given as an LTS M over a finite alphabet of events Σ, a finite set of regular predicates Φ and a regular abstraction $G = (\Sigma, Q, \delta, q_0)$ of M, i.e. $\mathcal{L}(M) \subseteq \mathcal{L}(G)$.*

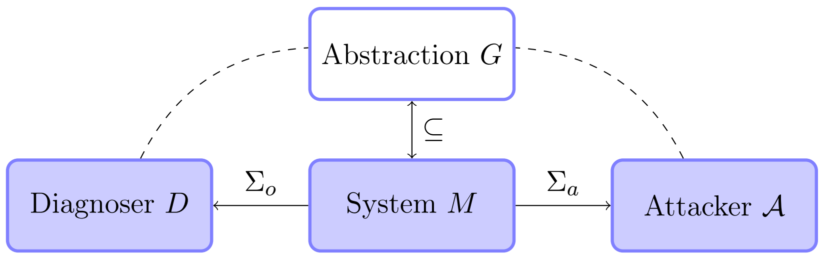


Figure 4.6: Diagnosing information flow using abstractions

- ***Problem:*** *Based on the abstraction G, compute a diagnoser D detecting every occurrence of secret information flow that may occur for an attacker reasonning from G.*

Given a set of regular secret predicates Φ, we can define the regular safety predicate ψ by

$$L(\psi) = \Sigma^* \setminus (\mathit{Disclose}(\mathcal{L}(G), \pi_a)(\Phi)\Sigma^*)$$

which consists in the set of words such that no secret information is disclosed to the attacker via a sound monitor based on G. In other words, we obtain the set of safe words by removing all the vulnerabilities revealed using G, following Proposition 4.14.

We can always verify whether $\mathcal{L}(G)$ is ψ-diagnosable since G is a finite LTS. As seen with Proposition 4.15, if $\mathcal{L}(G)$ is diagnosable, then so is $\mathcal{L}(M)$.

But we are not exactly interested in the diagnosability of $\mathcal{L}(M)$ since deciding whether a given word $w \in \mathcal{L}(M)$ belongs to $\mathit{Disclose}(\mathcal{L}(M), \pi_o)(\neg\psi)$ may be impossible. We need for this a notion of diagnosability defined with respect to both $\mathcal{L}(G)$ and $\mathcal{L}(M)$ and this is given by the following proposition.

Proposition 4.17 *Let $\mathcal{L}(G)$ be a regular abstraction of $\mathcal{L}(M)$ and ψ be a regular safety predicate. If $\mathcal{L}(G)$ is ψ-diagnosable, then there exists $N \in \mathbb{N}$ such that for all $w \in \mathcal{L}(M)$,*

$$w \in L(\neg\psi) \implies (\forall w' \in w^{-1}\mathcal{L}(M),\ |w'| \geq N \implies ww' \in \mathit{Disclose}(\mathcal{L}(G), \pi_o)(\neg\psi)) \tag{4.4}$$

Proof. Similar to Proposition 4.15. □

Let us now explain more the interest of Proposition 4.17 and suggest how it can be applied to detect vulnerabilities. The set of words $\mathit{Disclose}(\mathcal{L}(G), \pi_o)(\Phi)$ can be a set of false alarms but if some of them occur in $\mathcal{L}(M)$, then according to Proposition 4.14, they correspond to real cases of information flow. Then, the objective of the diagnoser is to detect whether the system M generates a word of $\mathit{Disclose}(\mathcal{L}(G), \pi_o)(\Phi)$. This is modeled by the regular safety predicate ψ defined by $L(\psi) = \Sigma^* \setminus (\mathit{Disclose}(\mathcal{L}(G), \pi_o)(\Phi)\Sigma^*)$ which is violated whenever the attacker could, at some time during the execution of M, infer secret information. But, thanks to proposition 4.17, if $\mathcal{L}(G)$ is ψ-diagnosable, then all occurrence of a real vulnerability revealed by G will eventually be detected by the diagnoser, after the occurrence of at most N events. This can be formalized by the following theorem which follows directly from Proposition 4.17.

Theorem 4.5 *If $\mathcal{L}(G)$ is ψ-diagnosable, then there exists $N \in \mathbb{N}$ such that for every word generated by M, i.e. $w \in \mathcal{L}(M)$, such that $w \in \mathit{Disclose}(\mathcal{L}(G), \pi_a)(\Phi)$,*

$$\forall w' \in w^{-1}\mathcal{L}(M),\ |w'| > N \implies ww' \in \mathit{Disclose}(\mathcal{L}(G), \pi_o)(\neg\psi)$$

A practical application of this is to define a complete and deterministic automaton A, accepting the language $\mathit{Disclose}(\mathcal{L}(G), \pi_o)(\neg\psi)$, applying the construction presented in Section 4.1, and implement the composition $A \parallel M$. The system M can be considered as secure as long as no execution of M generates a word accepted by A in $A \parallel M$.

4.5 Conclusion

In this chapter, we investigate two approaches to solve the opacity verification problem in the case of finite and infinite system. In the case of finite systems, we show that the opacity verification problem is PSPACE-complete. But the opacity verification problem is not decidable for infinite systems. For this case, we do not investigate sufficient conditions for opacity, like in [BKMR08] with the notion of uo-opacity. We propose a complementary approach as we focus on the detection of counterexamples to opacity, and more precisely on the detection of observed traces such that secret information is disclosed. We show how such a detection can be achieved on-line using a Galois connection based approach of abstract interpretation. We also show that combining overapproximation with underapproximation can help to statically, i.e. off-line, compute some counterexamples to opacity. We consider an other approach for the detection of counterexamples to opacity based on regular abstractions using the diagnosis theory.

It would be interesting to implement the theory presented in Section 4.3 and see if it can be applied to disclose secret information on programs written with a simple imperative programming language for example. Also, as an abstract interpretation framework can also be used to generate regular abstractions of a system, it would also be interesting to merge the techniques presented in Section 4.3 and Section 4.4 and apply the notion of diagnosis in this context. An other possible extension of this work of Section 4.4, can be to connect it with regular abstraction techniques like [LJJ06, LJ07] where regular abstractions are used to abstract the content of FIFO channels in communicating finite state machines.

5 Supervisory Control to Enforce Opacity

In this chapter, we investigate the problem of computing a controller enforcing opacity properties. We consider a system modeled a finite LTS M and confidential information given by a finite set of regular secret predicates Φ. This set of predicates will be fixed for the whole chapter, so opacity will implicitly mean opacity for the secret predicates Φ. As for the previous chapter, the attacker observes the events of $\Sigma_a \subseteq \Sigma$ with the projection denoted $\pi_a = \pi_{\Sigma \to \Sigma_a}$. As the secret predicates are trace-based, we will follow a language-based approach to study this control problem. In this chapter, we present some results about the existence and the effective computability of solutions to the opacity control problem. The complexity aspects of this problem are not investigated.

According to Ramadge and Wonham [RW87, RW89], the aim of supervisory control is to enforce a safety property on a transition system. This is achieved by computing a controller, given as an LTS C such that the controlled system, given by the parallel composition $C \parallel M$, does not violate the safety property. In order to model realistic situations, the controller cannot prevent the occurrence of some of the events. These events are called the uncontrollable events whereas the other ones are said controllable. It is also assumed that the controller only observes a subset of the events of M. Then, the control must then be performed under partial observation. This controller is generally expected to be as permissible as possible, in the sense that no unnecessary restriction should be imposed on the system. Other kinds of properties have been considered within this framework like for example the non-blocking property which requires that the control shall not prevent the system from eventually reaching some accepting states.

A potential application of supervisory control to enforce opacity is to automate some aspects of the implementation of secure systems. We consider a design process where the functional aspects are implemented first, for example by composing "off the shelf" components implementing basic operations. Then, if the security policy consists in integrity requirements, expressed by safety properties, and confidentiality requirements expressed by opacity properties, we can apply the theory presented in Chapter 4 to check if the resulting system meets these requirements. When this is not the case, we add an extra component, the controller, enforcing the security policy. We only consider the case of finite models in this chapter. The field of direct applications can be the design of some communications protocols like for example the dining cryptographer protocol, where a faithful finite LTS model can be provided. Opacity being general enough to express anonymity constraints, we expect that our approach can be applied to the design of such protocols. Also, like for model-checking, systems on chip can be a good target for the application of this theory, as there often exists a complete finite model of such systems. Then, for such practical applications, we are especially interested here in finite state controllers. Possible extensions of the following results to the case of infinite systems will be discussed in the conclusion of this chapter.

In the next section, we briefly present the supervisory control theory of Ramadge and Wonham (abbreviated R&W) and illustrate this theory by studying how to compute a controller enforcing safety properties. Then, we investigate the opacity control problem and show that the classical iteration of closure operators of R&W cannot always be applied for opacity. We address this problem with a new algorithmic approach to compute the most permissive controller enforcing opacity under the assumption that the alphabet of events observable by the controller is comparable with the set of events observable by the attacker.

5.1 The Supervisory Control Problem

The Supervisory Control Theory is a language based theory whose objective is to restrict the language generated by the system to a sublanguage satisfying a given property. We present here this theory by following a presentation that will be more suitable for its application to opacity. For a more complete presentation of supervisory control, the reader is referred for example to [CL08].

Let Ω be a family of languages defined over Σ. This set of languages represents the control objective, i.e. we search for a controller such that the controlled language is an element of Ω. This control is performed by composing the system M with a controller C, given as an LTS, such that $\mathcal{L}(C \parallel M)$ belongs to Ω. Such a controller will be called a *valid* controller for the objective Ω.

For example, if the objective is to enforce a trace-based safety predicate ψ defined by the prefix-closed language $L(\psi)$, then the objective is to compute a controller such that $\mathcal{L}(C \parallel M) \subseteq L(\psi)$, hence $\Omega = \mathcal{P}(L(\psi))$. If we want to enforce the liveness property on M, then a language L belongs to Ω if for all $w \in L$ and all runs $r \in \mathcal{R}(M)$ such that $tr(r) = w$, the restriction L does not prevent the system to proceed, i.e. there exists $\sigma \in w^{-1}L$ such that $\delta(\sigma, lst(r)) \neq \emptyset$, where δ is the transition function of M. For the opacity control problem, Ω is the set of opaque languages over Σ, i.e. $\Omega = \{L \subseteq \Sigma^* : \mathit{Disclose}(L, \pi_a)(\Phi) = \emptyset\}$. Finally, if we want to enforce the diagnosability of a safety regular predicate ψ then, following the definition of diagnosability 4.4, Ω is the set of languages L that are ψ-diagnosable.

The controller should also be as permissive as possible in the sense that no unnecessary restriction should be imposed on M. In this context, we say that C is a *supremal* valid controller if $\mathcal{L}(C \parallel M) \in \Omega$ and for every other controller, C', if $\mathcal{L}(C \parallel M) \subsetneq \mathcal{L}(C' \parallel M)$ then $\mathcal{L}(C' \parallel M) \notin \Omega$.

We assume that the occurrence of some events of Σ cannot be prevented by control. For example, if we want to enforce the confidentiality of data in an e-banking web service, we cannot prevent an attacker to send login requests to the service as Internet is an open network. All we can do is to deny access when no trustful guaranties, like a correct password or a known IP address, are provided. Also, we can disable for example some other output events of the web service to avoid data leakage towards an attacker eavesdropping the network traffic. We denote then by Σ_{uc} this set of uncontrollable events and the controllable ones are denoted Σ_c. Also, we can imagine that a controller, implemented beside the web service, may not be aware of some action like internal database access for example. This is modeled by assuming that only a subset Σ_o of the events of Σ are observable by the controller, implying that the controller must be such that $\mathcal{L}(C) \subseteq \Sigma_o^*$. Applying Proposition 2.3, the language of M controlled by C is $\mathcal{L}(C \parallel M) = \pi_o^{-1}(\mathcal{L}(C)) \cap \mathcal{L}(M)$ where $\pi_o = \pi_{\Sigma \to \Sigma_o}$. This implies that the occurrence of the events of $\Sigma_{uo} = \Sigma \setminus \Sigma_o$ cannot be prevented by the parallel composition. Therefore, $\Sigma_{uo} \subseteq \Sigma_{uc}$ or, equivalently, $\Sigma_c \subseteq \Sigma_o$[1]. The search space for possible controls over M is then determined by the two subsets Σ_c and Σ_o.

5.1.1 Language Based Approach for the Supervisory Control Problem

Following a language based approach and given a control objective Ω, the supervisory control problem consists then in finding a supremal sublanguage K of $\mathcal{L}(M)$ such that $K \in \Omega$ and $K = \mathcal{L}(C \parallel M)$ for some controller C. The system M being finite, there will exist a finite state controller C as soon as the language K is regular. Then, such a language K must be non-empty (contains at least ϵ), prefix-closed and satisfy two additional properties defined below.

First, given a controller C and a word $w \in \mathcal{L}(C \parallel M)$, if for $w' \in \mathcal{L}(M)$, $\pi_o(w) = \pi_o(w')$, then the word w' also belongs to $\mathcal{L}(C \parallel M) = \pi_o^{-1}(\mathcal{L}(C)) \cap \mathcal{L}(M)$. Hence, for a sublanguage K to be generated by controlling M under partial observation, K must be exactly recovered from its projection $\pi_o(K)$ and $\mathcal{L}(M)$. This is formalized by the notion of normality.

Definition 5.1 (Normality [RW87]) *A language K is normal w.r.t. $\mathcal{L}(M)$ and Σ_o if $\pi_o^{-1}(\pi_o(K)) \cap \mathcal{L}(M) \subseteq K$.*

Note that the union of an arbitrary number of normal languages is normal and this will be important in the sequel. This implies that given $L \subseteq \mathcal{L}(M)$, there always exists a supremal sublanguage $K \subseteq L$ such that K is normal. Such a language K is given by the union of all normal sublanguages of L.

[1]Note that in the R&W theory, it is generally not assumed that Σ_o and Σ_c are comparable

Second, we have seen that the uncontrollable events cannot be disabled by control. This means that for a language K, candidate to be a controlled sublanguage of $\mathcal{L}(M)$, for all $w \in K$ and all $\sigma \in \Sigma_{uc}$ such that $w\sigma \in \mathcal{L}(M)$, we must also have $w\sigma \in K$ since no controller can disable this event σ. This is formalized by the notion of controllability.

Definition 5.2 (Controllability [RW87]) *A language $K \subseteq \mathcal{L}(M)$ is controllable w.r.t $\mathcal{L}(M)$ and Σ_c if $K\Sigma_{uc} \cap \mathcal{L}(M) \subseteq K$.*

Note that the union of an arbitrary number of controllable languages is also controllable.

A *controlled* language is the outcome of the composition $\mathcal{L}(C \parallel M)$ where C is such that $\mathcal{L}(C) \subseteq \Sigma_o^*$ and does not prevent the occurrence of uncontrollable events. Therefore, a controlled language is a non-empty, prefix-closed, normal and controllable sublanguage of $\mathcal{L}(M)$.

Proposition 5.1 *If K is a controlled language, then $\Sigma_{uc}^* \cap \mathcal{L}(M) \subseteq K$. Moreover, $\Sigma_{uc}^* \cap \mathcal{L}(M)$ is the least controlled language.*

Proof. Let $K_0 = \Sigma_{uc}^* \cap \mathcal{L}(M)$ and let K be a controlled language. As K is non-empty and prefix-closed, $\epsilon \in K$. Then, if $\sigma \in \Sigma_{uc}$ such that $\sigma \in \mathcal{L}(M)$, then $\sigma \in K$ as K is controllable. It follows by induction that $K_0 \subseteq K$. To prove the normality, let $w \in K_0$. As $\Sigma_c \subseteq \Sigma_o$, $[w]_o \cap \mathcal{L}(M) \subseteq \pi_c^{-1}(\pi_c(w)) \cap \mathcal{L}(M) \subseteq K_0$. So K_0 is normal. Since K_0 is also non-empty, prefix-closed, K_0 is then the least controlled language included in $\mathcal{L}(M)$. $\square$

In the sequel, we will need to compute the supremal controlled language included in a sublanguage of $\mathcal{L}(M)$. The following proposition states necessary and sufficient conditions for its existence.

Proposition 5.2 *If $L \subseteq \mathcal{L}(M)$ then there exists a supremal controlled language $K \subseteq L$ if and only if $\Sigma_{uc}^* \cap \mathcal{L}(M) \subseteq L$.*

Proof. Again, let $K_0 = \Sigma_{uc}^* \cap \mathcal{L}(M)$. Applying Proposition 5.1, if there exists a controlled language $K \subseteq L$, then $K_0 \subseteq K \subseteq L$. Now, let $\mathcal{K}$ be the set of all controlled languages included in L. If $K_0 \subseteq L$ then $\mathcal{K}$ is not empty. Let $K = \cup\mathcal{K}$. As prefix-closeness is also closed under arbitrary union, this language K is then non-empty, prefix-closed, normal and controllable and is then the supremal controlled language included in L. $\square$

Remark 5.1 *The set of regular languages over an alphabet Σ is closed under finite union but not under arbitrary union. Indeed, let $\Sigma = \{a, b\}$ and the family of languages $\{L_n\}_{n\in\mathbb{N}}$*

defined by $L_n = a^n b^n$. The language $\cup\{L_n : n \in \mathbb{N}\}$ has an infinite set of residual languages and is therefore not regular. So, in the proof of Proposition 5.2, considering only regular languages in $\mathcal{K}$ will not imply that $K = \cup\mathcal{K}$ is regular.

Will now use the same ideas than for the proof of Proposition 5.2 to establish sufficient conditions for the existence of a supremal controlled language enforcing a property given by a family of languages Ω.

Theorem 5.1 *Let Ω be a control objective that we want to enforce on the system M. If the family of languages Ω is closed under arbitrary union, then either there exists no controller enforcing Ω or there exists a unique supremal prefix-closed, normal and controllable sublanguage $K \subseteq \mathcal{L}(M)$ such that $K \in \Omega$.*

Proof. Let $\mathcal{K}$ be the set of controlled languages $K \subseteq \mathcal{L}(M)$ such that $K \in \Omega$. If $\mathcal{K}$ is empty, then there exists no controller enforcing the objective Ω. Suppose then that $\mathcal{K} \neq \emptyset$ and let $K^\dagger = \cup\mathcal{K}$. Then $K^\dagger$ also belongs to Ω and we have seen that this language is also prefix-closed, normal and controllable. Then, this language $K^\dagger$ is the unique supremal controlled language enforcing the objective Ω. □

According to this theorem, the fact that the language $\Sigma_{uc}^* \cap \mathcal{L}(M)$ also belongs to Ω is a sufficient condition for the existence of a controlled language enforcing the property Ω. Indeed, it implies that the family of languages $\mathcal{K}$ is not empty as $\Sigma_{uc}^* \cap \mathcal{L}(M) \in \mathcal{K}$. For example, given a regular safety predicate ψ, the family of sublanguages of $L(\psi)$ is closed under union. Therefore there exists a supremal controller enforcing ψ if $\Sigma_{uc}^* \cap \mathcal{L}(M) \subseteq L(\psi)$. Also, it is clear that the family of languages enforcing the liveness of M is closed under union. So it follows directly from Theorem 5.1 that if there exists a controlled sublanguage of $\mathcal{L}(M)$ enforcing the liveness of M, there exists a supremal one. We will see in the next section that this also holds for opacity. But this is not always true for every kind of control objective. Indeed, according to Remark 4.9, the diagnosability is not preserved by union. Hence, Theorem 5.1 cannot be applied to prove the existence of a unique supremal controlled language that is diagnosable.

5.1.2 The Fixpoint Iteration Techniques

We will now present in a general framework the classical approach of R&W for solving control problems, i.e. deciding the existence and computing the supremal controlled language enforcing a given property as described above. This methodology consists in defining closure operators mapping a regular language to its largest sublanguage satisfying a given property, e.g. safety, liveness, controllability, etc. This closure operator must preserve regularity and prefix-closeness. Then, iterating this operator until a fixpoint is reached gives the researched supremal controlled sublanguage.

We start by introducing the operator $CN : \mathcal{P}(\mathcal{L}(M)) \to \mathcal{P}(\mathcal{L}(M))$ mapping a prefix-closed language to its supremal normal and controllable sublanguage. Let $L \subseteq \mathcal{L}(M)$ be a prefix-closed language and let $G = (\Sigma, Q, \delta, q_0, Q_f)$ be a finite automaton such that $\mathcal{L}(G) = \mathcal{L}(M)$ and $\mathcal{L}(G, q_0, Q_f) = L$. We compute $det_o(G) = (\Sigma_o, \mathcal{P}(Q), \Delta, X_0, Bad)$ where the set of accepting states Bad is defined by:

- defining first $F = \{X \subseteq \mathcal{P}(Q) : X \cap (Q \setminus Q_f) \neq \emptyset\}$;
- and then, $Bad = coreach_{det_o(G)}(\Sigma_o \setminus \Sigma_c)(F)$, the set of states of $det_o(G)$ such that a sequence of uncontrollable events leads to a state of F.

Let $\widetilde{L} = \mathcal{L}(det_o(G), X_0, \mathcal{P}(Q) \setminus Bad)$. Then, we define $CN(L) = \mathcal{L}(M) \cap \pi_o^{-1}(\widetilde{L})$.

Proposition 5.3 *The map $CN : \mathcal{P}(\mathcal{L}(M)) \to \mathcal{P}(\mathcal{L}(M))$ is a lower closure operator over the prefix-closed sublanguages of $\mathcal{L}(M)$. More precisely, given a prefix-closed language $L \subseteq \mathcal{L}(M)$, $CN(L)$ is the supremal normal and controllable sublanguage included in L. The operator CN also preserves regularity.*

Proof. Let $H = CN(L)$.

- If $L = \mathcal{L}(G, q_0, Q_f)$ is prefix-closed then it follows from the definition of F that $\mathcal{L}(det_o(G), X_0, F)$ is also prefix-closed. Then $\widetilde{L}$ and H are prefix-closed. It is also clear that H is regular as soon as L is regular.
- According to its definition, H is clearly normal.
- H is also controllable. Let $w \in H$ and $\sigma \in \Sigma_{uc}$ such that $w\sigma \in \mathcal{L}(M)$. If $\sigma \in \Sigma_{uo}$ then $w\sigma \in H$. Suppose now that $\sigma \in \Sigma_o$. Note that $\widetilde{L}(\Sigma_o \setminus \Sigma_c) \cap \pi_o(\mathcal{L}(M)) \subseteq \widetilde{L}$ by definition of Bad. So $\pi_o(w)\sigma \in \widetilde{L}$. As $w\sigma \in \mathcal{L}(M)$, it follows that $w\sigma \in H$.
- H contains every prefix-closed, normal and controllable sublanguage of L. Indeed let $L' \subseteq L$ be a normal and controllable sublanguage of L. Let $w \in L'$ and let $\mu = \pi_o(w)$. Let $w' \in \pi_o^{-1}(\mu) \cap \mathcal{L}(M)$. As L' is normal, $w' \in L' \subseteq L$ and if $q_0 \xrightarrow{w'} q$ is a run of G then $q \notin Q_f$. So $\Delta(\mu, X_0) \notin F$. L' being also controllable, if $u \in \Sigma_{uc}^*$ such that $wu \in \mathcal{L}(M)$, then $wu \in L'$. Applying the same arguments as above of for the word $wu \in L'$, $\Delta(\pi_o(wu), X_0) = \Delta(\pi_o(u), \Delta(\mu, X_0)) \notin F$. So $\Delta(\mu, X_0) \notin Bad$, which means that $\mu \in \widetilde{L}$ and then $w \in H$.

The language H is then the supremal prefix-closed, normal and controllable sublanguage of L. The operator CN is by consequence reductive, monotone and idempotent, i.e. is a lower closure operator over the prefix-closed sublanguages of $\mathcal{L}(M)$. $\square$

Suppose now that we want to enforce a property defined by Ω that is closed under arbitrary union. Consider the set $\mathcal{K}$ of controlled sublanguages $K \subseteq \mathcal{L}(M)$ such that $K \in \Omega$. According to Theorem 5.1, if $\mathcal{K} \neq \emptyset$, there exists a unique supremal controlled language enforcing Ω, given by $K^\dagger = \cup\mathcal{K}$. It remains to prove that this language $K^\dagger$ is effectively computable and regular.

For this, consider the operator $P : \mathcal{P}(\mathcal{L}(M)) \rightarrow \mathcal{P}(\mathcal{L}(M))$ mapping a sublanguage L to its supremal sublanguage K such that $K \in \Omega$. We assume that this operator a lower closure operator over the prefix-closed sublanguages of $\mathcal{L}(M)$ and preserves regularity.

Consider now the operator $k = CN \circ P$. This operator is monotone within the complete lattice of the prefix-closed sublanguages of $\mathcal{L}(M)$. So applying Theorem 2.2, k admits a unique greatest fixpoint $gfp(k)$. The language $gfp(k)$ is also a fixpoint for P. Indeed, $P(gfp(k)) \subseteq gfp(k)$. But if $P(gfp(k)) \subsetneq gfp(k)$ was true then, as CN is reductive, we would also have $CN \circ P(gfp(k)) \subsetneq gfp(k)$, which is not possible. This also implies that $CN(gfp(k)) = CN \circ P(gfp(k)) = gfp(k)$, so $gfp(k)$ is also a fixpoint for CN[2]. This is helpful to prove the following theorem.

Theorem 5.2 *If the family of languages Ω is closed under arbitrary union, then the greatest fixpoint $gfp(k)$ is the supremal controlled sublanguage of $\mathcal{L}(M)$ enforcing Ω. i.e.*

$$gfp(k) = K^\dagger$$

Proof. As $gfp(k)$ is a fixpoint for CN and P, $gfp(k) \in \mathcal{K}$ and so $gfp(k) \subseteq K^\dagger$. For the other inclusion, note that $K^\dagger$ is a fixpoint for CN as $K^\dagger$ is prefix-closed normal and controllable. The language $K^\dagger$ is also a fixpoint for P as $K^\dagger \in \Omega$ and is prefix-closed. So $K^\dagger$ is a fixpoint for k and then $K^\dagger \subseteq gfp(k)$. $\square$

Even though not presented in this way in the literature, this is the basis of the classical R&W methodology for establishing a proof that a supremal controlled language enforcing Ω can be effectively computed and is regular. When the family of target languages Ω is closed under arbitrary union, this methodology consists then in defining an operator P satisfying the property given above and iterating the computation of CN and P starting from $\mathcal{L}(M)$ until a fixpoint is reached. If such a fixpoint is reached after a finite number of iterations, then one obtains the supremal controlled language enforcing Ω and this language is regular as CN and P are both preserving regularity.

This methodology is successfully applied in the context of centralized or distributed con-

[2]Note that this also implies that $gfp(k) = gfp(P \circ CN)$.

trol to enforce different kinds of properties like for example, safety, liveness, non-blocking, etc.

For an illustration, we now apply this methodology to the simple case of safety properties, with the objective of enforcing integrity requirements in mind. In the next section, we apply will this control framework to enforce opacity properties.

5.1.3 The Safety Control Problem

Suppose that a security policy consists in a set of integrity requirements given by a set of regular safety predicates Ψ. The objective is then to compute a supremal controlled sublanguage $K \subseteq \mathcal{L}(M)$, such that for every $\psi' \in \Psi$, $K \subseteq L(\psi')$. So, we can without lost of generality consider the case of only one regular safety predicate, defining ψ by $L(\psi) = \cap\{L(\psi') : \psi' \in \Psi\}$[3]. The problem is then to compute, if existing, a controlled language K enforcing ψ on M. The existence of such a controlled language is due to the following proposition.

Proposition 5.4 *There exists a supremal controlled sublanguage enforcing the safety of ψ if and only if $\mathcal{L}(M) \cap \Sigma_{uc}^* \subseteq L(\psi)$.*

Proof. Let $K_0 = \Sigma_{uc}^* \cap \mathcal{L}(M)$. We have seen that $K_0 \subseteq L(\psi)$ is a sufficient condition as it proves that the set of controlled languages enforcing the safety of ψ is not empty. Suppose now that there exists a controlled language K enforcing ψ. We have seen that we must have $K_0 \subseteq K$. If $K_0 \subsetneq L(\psi)$ then $K \subsetneq L(\psi)$ which contradicts the existence of such K. So $K_0 \subseteq L(\psi)$ is a necessary condition. □

Assuming that $\mathcal{L}(M) \cap \Sigma_{uc}^* \subseteq L(\psi)$, to apply the fixpoint computation presented above, consider

$$\begin{array}{rccc} Safe: & \mathcal{P}(\mathcal{L}(M)) & \rightarrow & \mathcal{P}(\mathcal{L}(M)) \\ & L & \mapsto & L \cap L(\psi) \end{array}$$

It is clear that this operator maps every sublanguage of $\mathcal{L}(M)$ to its largest safe subset. Also, if L is prefix-closed and regular, then so is $Safe(L)$.

Proposition 5.5 *$CN \circ Safe(\mathcal{L}(M))$ is the supremal controlled sublanguage of $\mathcal{L}(M)$ enforcing the safety property ψ.*

Proof. Remark first that if $L \subseteq L(\psi)$, then $Safe(L) = L$. Since the operators CN and $Safe$ are reductive and $Safe(\mathcal{L}(M)) \subseteq L(\psi)$, then iteration of $CN \circ Safe$ are actually the iteration of CN starting from $Safe(\mathcal{L}(M))$. Finally, $gfp(CN \circ Safe) = CN \circ$

[3]Note that this simplification is not possible for a set of secret predicates

$Safe(\mathcal{L}(M))$ since CN is idempotent. Applying Theorem 5.2 establishes the proposition.

$\square$

Corollary 5.1 *If $\mathcal{L}(M) \cap \Sigma_{uc}^* \subseteq L(\psi)$, there exists a supremal controlled language enforcing the safety predicate ψ and this language is effectively computable and regular.*

Remark 5.2 *Note that we can easily generalize this result. Indeed, one particularity of the safety control problem is that the set of a safe languages is closed by inclusion.*

Given a control objective Ω that is closed by inclusion and a closure operator P mapping a language L to a language $L' \subseteq L$ such that $L' \in \Omega$, then $CN \circ P(\mathcal{L}(M)) \in \Omega$ and is therefore a solution to the control problem. For example, according to Proposition 4.15, the diagnosability is closed by inclusion. So, if Ω is the set of languages that are ψ-diagnosable for a safety predicate ψ, then by defining such an operator P, we can obtain with $CN \circ P(\mathcal{L}(M))$ a solution to the diagnosis control problem.

5.2 The Opacity Control Problem

In this section, our purpose is to solve the opacity control problem: computing a controller C such that $\mathcal{L}(C \parallel M)$ is Φ-opaque for π_a. To solve this problem, it is not sufficient to remove from $\mathcal{L}(M)$ all the words disclosing a secret. Indeed, we have assumed in the preceding chapter that the attacker knows the complete model of the system. In the present context, the implemented model is actually $C \parallel M$, and we have seen with the remark 4.7 that opacity may not be preserved by inclusion. More precisely, a controller removing information flow vulnerabilities and their extensions, i.e. such that $\mathcal{L}(C \parallel M) = \mathcal{L}(M) \setminus (Disclose(\mathcal{L}(M), \pi_a)(\Phi)\Sigma^*)$, may fail to ensure the opacity of $\mathcal{L}(C \parallel M)$. The following simple example illustrate this aspect.

Example 5.1 *Consider the language of the LTS depicted in Figure 5.1 where the secret predicate ϕ is defined by the set of words reaching the square states. Also, $\Sigma_a = \{a, b\}$, $\Sigma_c = \{c_1, c_2\}$ and $\Sigma_o = \Sigma = \{a, b, c_1, c_2\}$. The set of disclosing words of $\mathcal{L}(M)$ is $\{ac_1a\}$.*

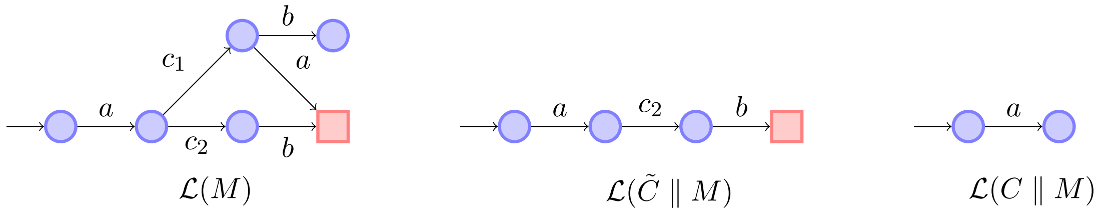

a
c_1
b
a
c_2
b
$\mathcal{L}(M)$
a
c_2
b
$\mathcal{L}(\tilde{C} \parallel M)$
a
$\mathcal{L}(C \parallel M)$

Figure 5.1: Removing disclosing words may introduce new vulnerabilities

So the controller $\tilde{C}$ is such that $\mathcal{L}(\tilde{C} \parallel M)$ is the supremal controlled language included *in $\mathcal{L}(M) \setminus (\{ac_1a\}\Sigma^*)$. But we can see on the figure that this controller does not enforce opacity as the words ac_2b now disclose ϕ in $\mathcal{L}(\tilde{C} \parallel M)$. The solution for this opacity control problem is the controller C disabling both c_1 and c_2 after observing a.*

Thus, as we suppose that the attacker knows the obtained controlled language, the exact control problem for opacity properties is to compute a supremal controller C such that the obtained language $\mathcal{L}(C \parallel M)$ is opaque.

5.2.1 Characterization of the Solution

Next, we give a set characterization of the solution to the opacity control problem and we investigate the existence of a supremal solution to this problem. To do so, we denote by $\Omega = \{L \subseteq \Sigma^* : Disclose(L, \pi_a)(\varPhi) = \emptyset\}$ the set of languages of $\mathcal{P}(\Sigma^*)$ that are $\varPhi$-opaque for π_a. We also consider the set

$$\mathcal{K} = \{K \subseteq \mathcal{L}(M) : K \text{ is a controlled language and } K \in \Omega\} \tag{5.1}$$

the set of controlled sublanguages of $\mathcal{L}(M)$ that are $\varPhi$-opaque for π_a. We also consider the prefix-closed language

$$K^\dagger = \cup\mathcal{K} \tag{5.2}$$

We know that $K^\dagger$ is prefix-closed, normal and controllable. The following proposition entails that $K^\dagger$ is also opaque.

Proposition 5.6 *Let L be a language and let $\mathcal{H}$ be a family of opaque sublanguages of L, then $\cup\mathcal{H}$ is also opaque.*

Proof. Let $\phi \in \varPhi$. Let $H = \cup\mathcal{H}$ and $w \in H$. There exists $H' \in \mathcal{H}$ such that $w \in H'$. As H' is opaque, there exists $w' \in H' \setminus L(\phi)$ such that $\pi_a(w') \sim_a \pi_a(w)$. So $w' \in H$ and then $w \notin Disclose(H, \pi_a)(\phi)$. $\square$

If $\mathcal{K} \neq \emptyset$, the previous proposition entails the existence of a unique supremal language $K^\dagger \subseteq \mathcal{L}(M)$ enforcing the $\varPhi$-opacity for the projection π_a. We still have to examine whether this language is regular (or at least, to exhibit sufficient conditions for its regularity) and to provide an effective computation of this language. We can state this problem as follows.

Problem 5.1 (Opacity Control) *With $K^\dagger = \cup\mathcal{K}$ as defined above with expressions (5.1) and (5.2), to solve the opacity control problem, we need to:*

1. *decide whether $K^\dagger \neq \emptyset$;*

2. *decide when $K^\dagger$ is regular;*

3. *compute a finite LTS C such that $\mathcal{L}(C \parallel M) = K^\dagger$.*

Like for the safety control problem, applying Proposition 5.2, we know that the opacity of $\Sigma_{uc}^* \cap \mathcal{L}(M)$ is a sufficient condition for the existence of a solution to Problem 5.1, i.e. that $K^\dagger \neq \emptyset$, and this simply because $\mathcal{K}$ is then non-empty[4]. But, unlike for the safety control problem, the opacity of $\Sigma_{uc}^* \cap \mathcal{L}(M)$ is not a necessary condition for $K^\dagger \neq \emptyset$ as illustrated by the following remark.

Remark 5.3 *Consider the system M depicted in Figure 5.2 with $\Sigma = \{x, c, a\}$, $\Sigma_a = \{a\}$ and $\Sigma_c = \{c\}$. Let $L(\phi) = \Sigma^* x \Sigma^*$. We see that $\mathcal{L}(M) \cap \Sigma_{uc}^*$ is not opaque whereas $\mathcal{L}(M)$*

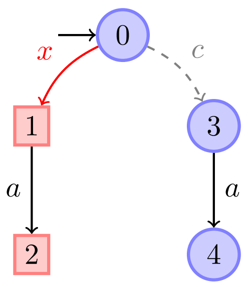


Figure 5.2: Unnecessary condition for the Opacity Control Problem

is opaque. Then $K^\dagger = \mathcal{L}(M)$!

5.2.2 An Operator for the Supremal Opaque Sublanguage

In order to apply the R&W fixpoint computation to prove the regularity and effective computability of $K^\dagger$, we now introduce the operator Op mapping a prefix-closed language to its supremal opaque sublanguage. To solve Problem 5.1, we investigate how to apply Theorem 5.2 and check whether the greatest fixpoint of $CN \circ Op$ can effectively be computed.

Define the map

$$\begin{array}{rccl} Op: & \mathcal{P}(\Sigma^*) & \rightarrow & \mathcal{P}(\Sigma^*) \\ & L & \mapsto & L \setminus (Disclose(L, \pi_a)(\Phi)\Sigma^*) \end{array}$$

Note that Op preserves the regularity, i.e. if L is regular, $Op(L)$ is also regular. Indeed, we can also write $Disclose(L, \pi_a)(\Phi) = \cup\{L \setminus \pi_a^{-1}(\pi_a(L \setminus L(\phi))) : \phi \in \Phi\}$, and since the operations π_a, π_a^{-1}, $\setminus$ and $\cup$ preserve the regularity, the language $Op(L) =$

[4]Recall that $\mathcal{K}$ is a set of non-empty languages

$L \setminus (\mathit{Disclose}(L, \pi_a)(\Phi)\Sigma^*))$ is also regular. The map *Op* also preserves the prefix-closeness as it is obtained by removing disclosing words and all their extensions.

Given a prefix-closed language L, we will see now that $\mathit{Op}(L)$ gives the supremal prefix-closed and opaque sublanguage of L [BBB+07]. The following technical lemma states that $\mathit{Op}(L)$ is an union of equivalence classes of L for π_a.

Lemma 5.1 *For all $w \in \mathit{Op}(L)$, $[w]_a \cap \mathit{Op}(L) = [w]_a \cap L$.*

Proof. Let $w \in \mathit{Op}(L)$. First, $[w]_a \cap \mathit{Op}(L) \subseteq [w]_a \cap L$ since $\mathit{Op}(L) \subseteq L$. Second, let $w' \in [w]_a \cap L$. Suppose that $w' \in \mathit{Disclose}(L, \pi_a)(\Phi)\Sigma^*$. Then, there exists $u' \leq w'$ such that $u' \in \mathit{Disclose}(L, \pi_a)(\phi)$ for one $\phi \in \Phi$. This can equivalently be reformulated as $\pi_a(u') \in \mathit{DTraces}(L, \pi_a)(\phi)$. But there exists $u \leq w$ such that $\pi_a(u) = \pi_a(u')$ and then $u' \in \mathit{Disclose}(L, \pi_a)(\phi)$. So, $w \notin \mathit{Op}(L)$ which is a contradiction. Hence, $w' \in [w]_a \cap \mathit{Op}(L)$, and the lemma is then established. $\square$

We show now that, as expected, the previous lemma implies that the language $\mathit{Op}(L)$ is opaque.

Lemma 5.2 *For every language $L \subseteq \Sigma^*$, $\mathit{Op}(L)$ is opaque.*

Proof. Let $w \in \mathit{Op}(L)$. If for all $w' \in [w]_a \cap \mathit{Op}(L)$, $w' \in L(\phi)$ then this also holds for all $w' \in [w]_a \cap L$ and then $w \in \mathit{Disclose}(L, \pi_a)(\phi)$. This contradicts that $w \in \mathit{Op}(L)$. We conclude then that *Op* is opaque. $\square$

Proposition 5.7 *The map Op is a lower closure operator over the sublattice of prefix-closed languages of $\mathcal{P}(\Sigma^*)$.*

Proof. It is clear that *Op* is reductive, i.e. that $\mathit{Op}(L) \subseteq L$ for all $L \subseteq \mathcal{L}(M)$. According to lemma 5.2, *Op* is idempotent since $\mathit{Disclose}(\mathit{Op}(L), \pi_a)(\Phi) = \emptyset$. As we have already seen that *Op* preserves prefix-closeness, it remains to show that *Op* is monotone. Let $L_1 \subseteq L_2$ be two prefix-closed languages. Let $w \in \mathit{Op}(L_1)$. Then $w \in L_2$. Suppose that there exists $u \leq w$ such that $u \in \mathit{Disclose}(L_2, \pi_a)(\phi)$ for one $\phi \in \Phi$. Then, as L_1 is prefix closed and w also belongs to L_1, $u \in L_1$. So, according to Proposition 4.14, $u \in \mathit{Disclose}(L_1, \pi_a)(\phi)$ which is not possible as $w \in \mathit{Op}(L_1)$. So, this implies that $w \in \mathit{Op}(L_2)$. Then, $\mathit{Op}(L_1) \subseteq \mathit{Op}(L_2)$ and then *Op* is monotone. We conclude that *Op* is a lower closure operator over the set of prefix-closed languages of Σ^*. $\square$

We will see now that the operator *Op* give the supremal prefix-closed and opaque sublanguage of a prefix-closed language L.

Proposition 5.8 *Given a prefix-closed language $L \subseteq \Sigma^*$, $Op(L)$ is the supremal opaque sublanguage of L.*

Proof. Let H be the union of all prefix-closed and opaque sublanguages of L. As $Op(L)$ is opaque and prefix-closed, $Op(L) \subseteq H$. Also $H \subseteq L$ so $Op(H) \subseteq Op(L)$ by monotony. According to Proposition 5.6, H is opaque and prefix-closed as the union of prefix-closed and opaque languages. Then, $Op(H) = H$ and $H \subseteq Op(L)$. Finally $Op(L) = H$. □

Note that $Op(L)$ can be empty. In the case when $Op(\mathcal{L}(M)) = \emptyset$, there is no way to enforce opacity on M by control.

5.3 Computation of the Supremal Controller when Σ_a and Σ_o are Comparable

In this section, we exhibit sufficient conditions such that the classical fixpoint computation of R&W provides a solution to Problem 5.1. These conditions depend on the relations between the set of controllable events Σ_c, the observable events Σ_o and the events of Σ_a which are observable by the attacker. We show that the fixpoint computation of $CN \circ Op$ terminates after a finite number of iteration when $\Sigma_o \subseteq \Sigma_a$ and when $\Sigma_a \subseteq \Sigma_c$. But we present an example of LTS, where $\Sigma_a \subseteq \Sigma_o$, such that the iterations of $CN \circ Op$ does not terminate after a finite number of iterations.

Define now the operator

$$k = CN \circ Op$$

According to the theory presented in section 5.1, the fixpoint iteration technique can be applied and, as a direct application of Theorem 5.2, we obtain

$$gfp(k) = K^\dagger$$

As $\mathcal{K} \neq \emptyset$ (since $\{\epsilon\} \in \mathcal{K}$), $gfp(k)$ is non-empty. We next study the sufficient conditions cited above under which $gfp(k)$ is regular and can effectively be computed.

5.3.1 The Case $\Sigma_o \subseteq \Sigma_a$

With the assumption that $\Sigma_o \subseteq \Sigma_a$, the controller observes and controls only a part of the events of the attacker[5], meaning that the controller is less accurate than the attacker

[5]Recall $\Sigma_c \subseteq \Sigma_o$

regarding the internal behavior of M. This is a sufficient condition to solve the control problem by computing $gfp(k)$.

Proposition 5.9 *If $\Sigma_o \subseteq \Sigma_a$, then $gfp(k) = k(\mathcal{L}(M))$. This language is then regular and effectively computable.*

Proof. Let $L_1 = Op(\mathcal{L}(M))$ and $K_1 = CN(L_1) = k(\mathcal{L}(M))$. Consider $w \in K_1 \cap L(\phi)$. As L_1 is opaque, there exists $w' \in L_1$ such that $w \sim_a w'$ and $w' \notin L(\phi)$ As $\Sigma_o \subseteq \Sigma_a$ and $w \sim_a w'$, we get $w \sim_o w'$. Hence, K_1 being normal, we also have $w' \in K_1$, which entails that K_1 is opaque. Hence, $K_1 = Op(\mathcal{L}(M))$. But K_1 is also a fixpoint for CN and then $k(K_1) = K_1$. So $K_1 = k(\mathcal{L}(M)) = gfp(k)$. □

Applying Proposition 5.9, we solve the items (2) and (3) of Problem 5.1 together. The construction of the operators CN and Op implicitly provides an LTS A such that $\mathcal{L}(A) = k(\mathcal{L}(M))$. Then, $C = det_o(A)$ is a possible solution for the controller. But of course, this approach may not be optimal. For example, if for a practical application, the number of states of C is critical (one can think of an application to electronic), then, algorithms to compute the minimal DFA accepting the language $\pi_o(k(\mathcal{L}(M)))$ would be of interest.

5.3.2 The Case $\Sigma_a \subseteq \Sigma_o$

In the context $\Sigma_a \subseteq \Sigma_o$, we show that solving the Opacity Control Problem under the assumption $\Sigma = \Sigma_o$ (full observation) induces a solution of the Opacity Control Problem to the general case $\Sigma_a \subseteq \Sigma_o$. We will apply this result to lighten the presentation for the subsequent developments. Especially, the parameter Σ_o of the Opacity Control Problem will therefore be eliminated in the rest of this chapter. This will also imply that every sublanguage of $\mathcal{L}(M)$ will be normal, and then, the notion of normality will be forgotten.

To prove that we can simplify the control problem in such a way, define

$$\mathcal{K} = \mathcal{F}(\Sigma, \mathcal{L}(M), \Phi, \Sigma_c, \Sigma_o, \Sigma_a)$$

where $\mathcal{K}$ is the set specified with expression (5.1) to characterize the set of solutions to the Opacity Control Problem. We similarly define the set

$$\mathcal{K}_o = \mathcal{F}_o(\Sigma_o, \pi_o(\mathcal{L}(M)), \Phi_o, \Sigma_c, \Sigma_o, \Sigma_a)$$

where Φ_o is the set of regular secret predicates defined by

$$\Phi_o = \{\phi_o \text{ defined by } L(\phi_o) = \pi_o(L(\phi)) \setminus \pi_o(\mathcal{L}(M) \setminus L(\phi)),\ \phi \in \Phi\}$$

Note that for $\phi \in \Phi$, $L(\phi_o) \subseteq \pi_o(L(\phi))$. Intuitively, to define ϕ_o from ϕ, we use the fact that $\Sigma_a \subseteq \Sigma_o$ and remove from $\pi_o(L(\phi_o))$ the words (of Σ_o^*) that do not disclose the secret regarding the projection π_o. Indeed, according to Proposition 3.3, we already know that such words will not disclose secret information for the projection π_a. With this definition, we will prove that $L(\phi_o)$ exactly encodes the information that we need to conceal on $\pi_o(\mathcal{L}(M))$ to relate the set $\mathcal{K}$ and $\mathcal{K}_o$.

Proposition 5.10 *The Opacity Control Problem with the parameters* (Σ, $\mathcal{L}(M)$, Φ, Σ_c, Σ_o, Σ_a) *is equivalent to the same problem with the parameters* (Σ_o, $\pi_o(\mathcal{L}(M))$, Φ_o, Σ_c, Σ_o, Σ_a).

To lighten the presentation and the proof of this proposition, we first establish three intermediate lemmas.

Lemma 5.3 *For all* $K \in \mathcal{K}$, $\pi_o(K) \in \mathcal{K}_o$.

Proof. $K \subseteq \mathcal{L}(M) \Rightarrow \pi_o(K) \subseteq \pi_o(\mathcal{L}(M))$, and π_o also preserves the non-emptiness and prefix-closeness.

We show that $\pi_o(K)$ is controllable w.r.t. $\pi_o(\mathcal{L}(M))$ and Σ_c. Let $\nu \in \pi_o(K)$ and $\sigma \in \Sigma_o \setminus \Sigma_c$ such that $\nu\sigma \in \pi_o(\mathcal{L}(M))$. As $\mathcal{L}(M)$ is prefix-closed, $\nu = \pi_o(w)$ for some $w \in \mathcal{L}(M)$ such that $w\sigma \in \mathcal{L}(M)$. Let $\nu = \pi_o(v)$ for some $v \in K$. Then $\pi_o(v) = \pi_o(w)$. As K is normal w.r.t. $\mathcal{L}(M)$ and Σ_o, $w \in \mathcal{L}(M)$ and $v \in K$ implies $w \in K$. As $w\sigma \in \mathcal{L}(M)$ and $\sigma \notin \Sigma_c$, $w\sigma \in K$ by controllability of K. Therefore, $\nu\sigma \in \pi_o(K)$ as required.

The projected language $\pi_o(K)$ is certainly normal w.r.t. $\pi_o(\mathcal{L}(M))$ and Σ_o, because $\pi_o(K) \subseteq \pi_o(\mathcal{L}(M)) \subseteq \Sigma_o^*$.

We show finally that $\pi_o(K)$ is opaque. Let $\phi_o \in \Phi_o$ and $\nu \in L(\phi_o) \cap \pi_o(K)$. Then $\nu = \pi_o(w)$ for some $w \in K \subseteq \mathcal{L}(M)$ and by definition of $L(\phi_o)$, $w \in L(\phi)$. As K is opaque, $\pi_a(w) = \pi_a(w')$ for some $w' \in K \setminus L(\phi)$. Let $\nu' = \pi_o(w')$. So $\nu' \in \pi_o(K)$, $\pi_a(\nu) = \pi_a(w) = \pi_a(w') = \pi_a(\nu')$ and $\nu' \notin L(\phi_o)$ since $w' \in \pi_o^{-1}(\nu')$, $w' \in K \subseteq \mathcal{L}(M)$, and $w' \notin L(\phi)$. □

Lemma 5.4 *If* $K_o \in \mathcal{K}_o$, *then* $\pi_o^{-1}(K_o) \cap \mathcal{L}(M) \in \mathcal{K}$.

Proof. Let $K = \pi_o^{-1}(K_o) \cap \mathcal{L}(M)$. Then $K \subseteq \mathcal{L}(M)$, K is prefix-closed because K_o and $\mathcal{L}(M)$ are both prefix-closed, and $K \neq \emptyset$ because K_o is a non-empty subset of $\pi_o(\mathcal{L}(M))$.

We show that K is controllable w.r.t. $\mathcal{L}(M)$ and Σ_c. Let $w\sigma \in \mathcal{L}(M)$ with $w \in K$ and $\sigma \in \Sigma \setminus \Sigma_c$. If $\sigma \notin \Sigma_o$, then $w\sigma \in K$ because $\pi_o(w\sigma) = \pi_o(w) \in K_o$. Suppose now that $\sigma \in \Sigma_o$. Then $\nu = \pi_o(w)$ belongs to K_o, $\sigma \in \Sigma_o \setminus \Sigma_c$, and $\nu\sigma \in \pi_o(\mathcal{L}(M))$. As K_o is controllable w.r.t. $\pi_o(\mathcal{L}(M))$ and Σ_c, $\nu\sigma \in K_o$. Therefore, $w\sigma \in \pi_o^{-1}(K_o)$, and since $w\sigma \in \mathcal{L}(M)$, $w\sigma \in K$ as required.

It follows directly from the definition $K = \pi_o^{-1}(K_o) \cap \mathcal{L}(M)$ that K is normal w.r.t. $\mathcal{L}(M)$ and Σ_o.

We show finally that K is opaque. Let $\phi \in \Phi$. Let $w \in K \cap L(\phi)$ and let $\nu = \pi_o(w)$. Then $\nu \in K_o$, hence $\pi_o^{-1}(\nu) \cap \mathcal{L}(M) \subseteq K$.

- If $\pi_o^{-1}(\nu) \cap \mathcal{L}(M)$ is not included in $L(\phi)$, then $\pi_o(w) = \pi_o(w')$ for some $w' \in \pi_o^{-1}(\nu) \cap (K \setminus L(\phi))$. Thus $\pi_a(w') = \pi_a(w)$ and $w' \in K \setminus L(\phi)$.
- If $\pi_o^{-1}(\nu) \cap \mathcal{L}(M)$ is included in $L(\phi)$, then $\nu \in L(\phi_o)$ by definition of this set. As K_o is ϕ_o-opaque for π_a, $\pi_a(\nu) = \pi_a(\nu')$ for some $\nu' \in K_o \setminus L(\phi_o)$. By definition of $L(\phi_o)$, $\nu' = \pi_o(w')$ for some $w' \in \mathcal{L}(M) \setminus L(\phi)$. Now $w' \in \pi_o^{-1}(\nu')$ and $\nu' \in K_o$, hence $\pi_o^{-1}(\nu') \cap \mathcal{L}(M) \subseteq K$ by definition of K. Therefore, $w' \in K \setminus L(\phi)$. Finally, $\pi_a(w') = \pi_a(w)$ because $\pi_a(w) = \pi_a(\nu) = \pi_a(\nu') = \pi_a(w')$.

□

And the last lemma, whose proof is straightforward.

Lemma 5.5 *The maps π_o and $\pi_o^{-1}(\cdot) \cap \mathcal{L}(M)$ establish a Galois connection between the sublanguages of $\mathcal{L}(M)$ and the sublanguages of $\pi_o(\mathcal{L}(M))$, with $\pi_o \circ \pi_o^{-1}(\cdot) \cap \mathcal{L}(M) = id_{\pi_o(\mathcal{L}(M))}$.*

Note that both operations π_o and $\pi_o^{-1}(\cdot) \cap \mathcal{L}(M)$ are monotone as a consequence of Lemma 5.5. We can now start the proof of Proposition 5.10.

Proof.(of Proposition 5.10) In view of Lemmas 5.3 and 5.4, $\mathcal{K} \neq \emptyset$ if and only if $\mathcal{K}_o \neq \emptyset$. This implies the equivalence for the first item of Problem 5.1, i.e. the emptiness of the set of controlled language enforcing opacity.

Moreover, applying Lemma 5.5, the following relations hold for all $K \in \mathcal{K}$ and $K_o \in \mathcal{K}_o$,

- $\pi_o(K) \subseteq K_o \Rightarrow K \subseteq \pi_o^{-1}(K_o) \cap \mathcal{L}(M)$
- $K \subseteq \pi_o^{-1}(K_o) \cap \mathcal{L}(M) \Rightarrow \pi_o(K) \subseteq K_o$
- $K \subseteq \pi_o^{-1} \circ \pi_o(K) \cap \mathcal{L}(M)$
- $K_o = \pi_o(\pi_o^{-1}(K_o) \cap \mathcal{L}(M))$

One deduces then the following. Since ${K_o}^\dagger = \cup \mathcal{K}_o \in \mathcal{K}_o$, then for any $K \in \mathcal{K}$, $\pi_o(K) \subseteq {K_o}^\dagger$, so $K \subseteq \pi_o^{-1}({K_o}^\dagger) \cap \mathcal{L}(M)$. Hence $K^\dagger \subseteq \pi_o^{-1}({K_o}^\dagger) \cap \mathcal{L}(M)$ and then $\pi_o(K^\dagger) \subseteq K_o^\dagger$. Symmetrically, since $K^\dagger = \cup \mathcal{K} \in \mathcal{K}$ then for any $K_o \in \mathcal{K}_o$, $\pi_o^{-1}(K_o) \cap \mathcal{L}(M) \subseteq K^\dagger \subseteq \pi_o^{-1} \circ \pi_o(K^\dagger) \cap \mathcal{L}(M)$, so $\pi_o(\pi_o^{-1}(K_o) \cap \mathcal{L}(M)) = K_o \subseteq \pi_o(K^\dagger)$. Hence, $K_o^\dagger \subseteq \pi_o(K^\dagger)$. Finally, $K_o^\dagger = \pi_o(K^\dagger)$.

This also implies that $\pi_o^{-1}(K_o^\dagger) \cap \mathcal{L}(M) = \pi_o^{-1}(\pi_o(K^\dagger)) \cap \mathcal{L}(M)$. As $K^\dagger \subseteq \pi_o^{-1}(\pi_o(K^\dagger)) \cap \mathcal{L}(M)$, it follows that $K^\dagger = \pi_o^{-1}(K_o^\dagger) \cap \mathcal{L}(M)$.

The fact that both operators $\pi_o(\cdot)$ and $\pi_o^{-1}(\cdot) \cap \mathcal{L}(M)$ preserve regular languages and are effectively computable for regular languages concludes the proof for the equivalence of items (2) and (3) of Problem 5.1. □

Based on Proposition 5.10, whenever $\Sigma_a \subseteq \Sigma_o$, we can reformulate the opacity control problem in terms of the abstract system induced by the observation map π_o (with the ϵ-closure of M) and a new set of secret predicates Φ_o derived from Φ. Then, we can solve the problem in this abstract setting, and lift up the solution $K_o^\dagger$ to the original setting, as $K^\dagger = \pi_o^{-1}(K_o^\dagger) \cap \mathcal{L}(M)$.

In the sequel, we assume that $\Sigma_a \subseteq \Sigma_o$, so applying Proposition 5.10, we can consider without lost of generality that $\Sigma_o = \Sigma$. In that case, we will show that the existence of the optimal controller can always be decided and that this controller is regular.

The fixpoint computation when $\Sigma_a \subseteq \Sigma_c$

We present now the case when $\Sigma_a \subseteq \Sigma_c$. This means that the controller can observe all the events of the attacker and control them. With this assumption, we will prove that the fixpoint computation terminates.

Proposition 5.11 *If $\Sigma_a \subseteq \Sigma_c$, then $gfp(k) = Op(\mathcal{L}(M))$.*

Proof. We show that the prefix-closed language $Op(\mathcal{L}(M))$ is controllable, which implies that this language is a fixpoint for k[6]. Let $w \in Op(\mathcal{L}(M))$ and $\sigma \notin \Sigma_c$, such that $w\sigma \in \mathcal{L}(M)$. As $\Sigma_a \subseteq \Sigma_c$, $\sigma \notin \Sigma_a$ and then $w\sigma \in [w]_a \cap \mathcal{L}(M)$. According to Lemma 5.1, $w\sigma \in [w]_a \cap Op(\mathcal{L}(M))$, and then $w\sigma \in Op(\mathcal{L}(M))$. □

Remark 5.4 *It is a good place to make a link with the work of [TO08] that has been made in parallel of this thesis. In this article, the authors investigate the opacity control problem under full observation. With the assumption*

$$\forall w, w' \in \mathcal{L}(M),\ \forall \sigma \in \Sigma_{uc} \cap \Sigma_a,\ w \sim_a w' \wedge w\sigma \in \mathcal{L}(M) \implies w'\sigma \in \mathcal{L}(M)$$

that is more general than $\Sigma_a \subseteq \Sigma_c$, they show that the fixpoint computation of $CN \circ Op$ terminates after a finite number of iterations, which proves the regularity and the computability of the supremal controlled language. Note also That, thanks to Proposition 5.10, we can extend this work to the case $\Sigma_a \subseteq \Sigma_o$.

[6]Recall that there is no need to prove the normality in the case $\Sigma_o = \Sigma$

A more general case: Σ_a and Σ_c not comparable

We have seen different situations where the fixpoint computation of $CN \circ Op$ provided a solution to the opacity control problem. We will see that this methodology cannot always be applied for opacity properties, which makes the opacity control problem out of the scope of the classical R&W operator iteration techniques. We illustrate this point with a counterexample.

Example 5.2 *Consider the LTS M shown in Figure 5.3 where $\Sigma_a = \{a, b, X, Y\}$, $\Sigma_{ua} = \{c, u\}$, $\Sigma_c = \{c\}$ and $\Sigma_o = \Sigma$. The transitions that are unobservable to the attacker are pictured in gray and the controllable transitions with dashed arrows. The secret predicate is such that $L(\phi)$ is the set of the sequences reaching the states represented with squares in M. Let $K_i = k^i(\mathcal{L}(M))$ denote the language computed after i iterations of the operator $k = CN \circ Op$. The LTS $det_a(M)$ is depicted in Figure 5.4. Based on $det_a(M)$, we can see*

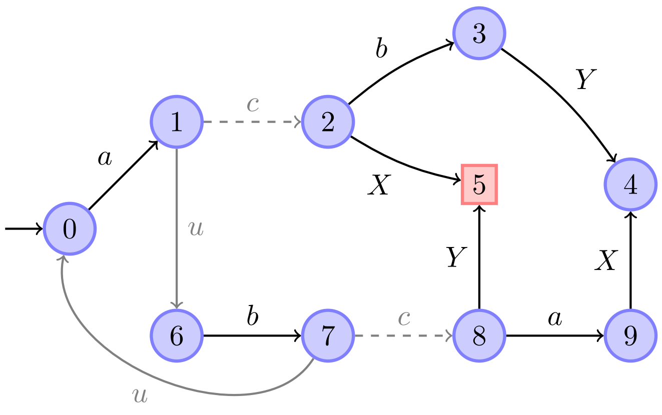


Figure 5.3: A problematic LTS M

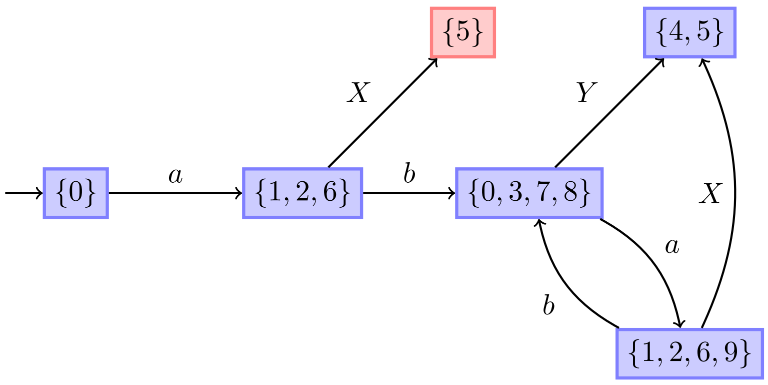


Figure 5.4: The LTS $det_a(M)$

that only the observed trace aX discloses the secret. The set of counterexamples to opacity is then $Disclose(\mathcal{L}(M), \pi_a)(\phi) = \pi_a^{-1}(aX) \cap \mathcal{L}(M) = \{acX\}$. Therefore, $Op(\mathcal{L}(M)) =$

$\mathcal{L}(M) \setminus \{acX\}$, so to restrict this language to a controllable sublanguage, we need to disable the occurrence of the event c after the first a. We obtain then $K_1 = CN \circ Op(\mathcal{L}(M))$. The LTS that generates K_1 is represented in Figure 5.5. In K_1, the word acX has disappeared

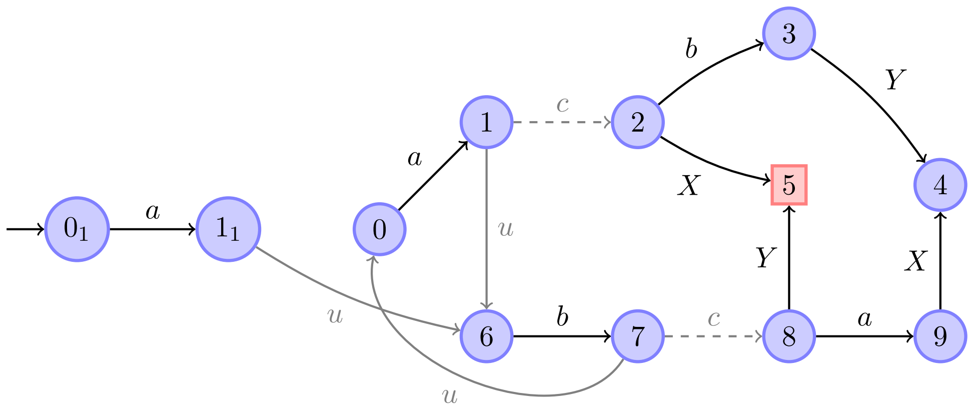


Figure 5.5: The language K_1

but doing so, we have introduced a new counterexample to opacity. Indeed, the word $aubcY$ now discloses the secret, which requires to disable the event c after aub. The resulting language K_2 is depicted in Figure 5.6. Now, the word $aubuacX$ discloses the secret in K_2.

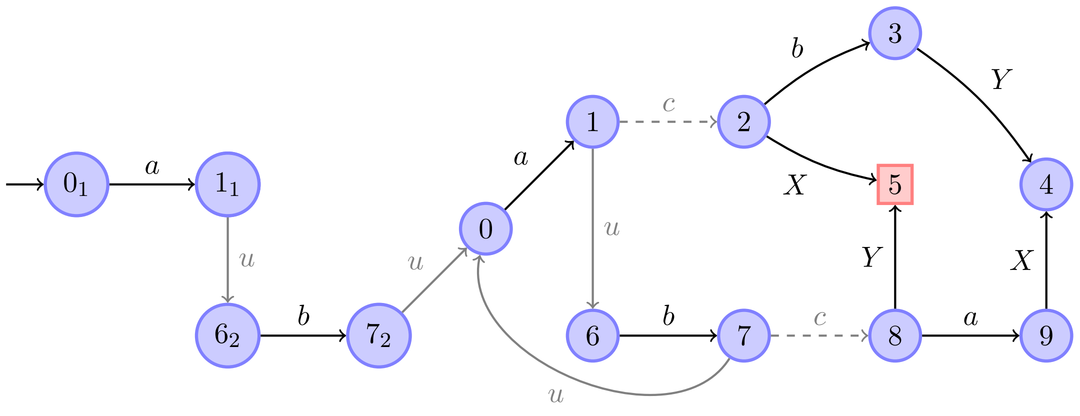


Figure 5.6: The language K_2

We see on the figure 5.6 that by iterating the operator Op and CN, we will always retrieve the same pattern on the right of the figure, after the states $0, 1, 2, \ldots$. On these iterations, we will always be able to find a single word disclosing ϕ, alternatively finishing by acX or bcY and ending at state 5. Even though the limit $gfp(k)$ of this decreasing chain is the regular language $(aubu)^*$, the fixpoint iteration produces then a strictly decreasing and infinite sequence of languages K_i showing that the above fixpoint computation algorithm does not terminate.

As the classical R&W fixpoint iteration technique cannot always be applied for opacity, we need to design a new algorithmic approach to solve the opacity control problem. The fixpoint computation does not provide a solution because it fails to find good invariants from the set of disclosing words. Indeed, we can remark on the example 5.2 that the words with suffixes acX or bcY will always be problematic whereas the iterations of k only remove prefixes of such words.

The algorithmic approach that we present in the sequel takes into account how the knowledge of the attacker influences the way to control. More precisely, we show that this control only depends on the state of the system and the set of states the system may have reached from the point of view of the attacker[7].

Regularity of the Supremal Controlled Language

Next, we analyze the influence of opacity on the control law, and we show that this control law can be represented in a "state based" fashion. With this analysis, we will be able to prove that the supremal controlled language is always regular under the assumption that $\Sigma_o = \Sigma$ (or equivalently $\Sigma_o \subseteq \Sigma$). This will also suggest a structure on which the controller can effectively be computed.

In order to avoid considering the case when $\mathcal{K} = \emptyset$, we now make some assumptions about Φ and M. This will simplify the presentation in the rest of this chapter as the set $\mathcal{K}$ will always be non-empty. We assume that:

1. $\epsilon \notin L(\phi)$ for all $\phi \in \Phi$;
2. the transitions starting from the initial states of M are controllable and observable by the attacker.

With this assumptions, $\{\epsilon\}$ is non-empty, prefix-closed, controllable and normal. Also, $\{\epsilon\}$ is opaque so $\{\epsilon\} \in \mathcal{K}$. Therefore $\mathcal{K} \neq \emptyset$ and $K^\dagger = \cup\mathcal{K}$ is the unique supremal controlled language enforcing the opacity on M.

Remark 5.5 *This two assumptions can be made without lost of generality. Indeed, we can consider a new state q', a new event σ' and replace M by a new LTS M' with initial state q' such that there is a transition labeled by σ' from q' to every initial state of M. Then, we consider the opacity control problem for M', the set of regular secret predicates Φ' defined by $L(\phi') = \sigma' L(\phi)$ for every $\phi \in \Phi$ and the new set of events $\Sigma_c' = \Sigma_c \cup \{\sigma'\}$ (then $\Sigma_o' = \Sigma_o \cup \{\sigma'\}$) and $\Sigma_a' = \Sigma_a \cup \{\sigma'\}$. Define $\mathcal{K}'$ similarly to the expression (5.1). With this construction, $K' \in \mathcal{K}'$ if and only if $\sigma'^{-1}K' \in \mathcal{K}$ which means that the problem 5.1(2) and (3) are equivalent for M and M'. Then $K^\dagger = \{\epsilon\}$ if and only if $\mathcal{K} = \emptyset$.*

[7]Intuitively, the state estimates from $det_a(M)$

To simplify the notations, we will also note $L_0 = \mathcal{L}(M)$ for the rest of this chapter. In the sequel, it will also be easier to manipulate a deterministic LTS. Therefore, applying Proposition 3.7, let $G = (\Sigma, Q, \delta, q_0)$ be a deterministic LTS such that $\mathcal{L}(G) = L_0$ and for all $\phi \in \Phi$, there exists $F(\phi) \subseteq Q$ such that for all $w \in \mathcal{L}(G)$,

$$\delta(w, q_0) \in F(\phi) \iff w \in L(\phi)$$

According to the hypothesis given above, $q_0 \notin F(\phi)$, for all $\phi \in \Phi$ and $\delta(\cdot, q_0)^{-1}(Q) \subseteq \Sigma_c \cap \Sigma_a$.

We reasoned until now upon the family $\mathcal{K}$ of all non-empty and prefix-closed sublanguages K of L_0 such that K is controllable and opaque. We will now replace $\mathcal{K}$ with a family $\mathcal{H}$ of maps $h : \mathcal{P}(Q) \times Q \to \mathcal{P}(\Sigma^*)$ each of them defining a doubly indexed collection of languages $h(e, q) \subseteq l(q)$, where $l : Q \to \mathcal{P}(\Sigma^*)$ is the map defined by $q \mapsto \mathcal{L}(G, q, Q)$.

Definition 5.3 *Let $\mathcal{H}$ be the family of maps $h : \mathcal{P}(Q) \times Q \to \mathcal{P}(\Sigma^*)$ such that the following conditions are satisfied for all $(e, q) \in \mathcal{P}(Q) \times Q$:*

1. *$h(e, q)$ is a prefix-closed and controllable sublanguage of $l(q)$,*
2. *$\forall \phi \in \Phi,\ \forall w \in h(e, q),\ \exists q' \in e,\ \exists w' \in h(e, q'),\ w \sim_a w' \wedge \delta(w', q') \notin F(\phi)$.*

Intuitively, if a sequence w of a controlled language has reached the state $q = \delta(w, q_0)$, and the attacker thinks that G is in one of the states of e according to the observation $\pi_a(w)$[8], then $h(e, q)$ represents the set of future words accepted by the controlled language. The second item of Definition 5.3 formalize the constraints such controlled language should satisfy in these future words in order to enforce opacity.

Note that it is not required that the languages $h(e, q)$ are regular. Also, to avoid dealing with partial function, we assume that $h(e, q)$ can be empty. In the case of empty languages, the items (1) and (2) of Definition 5.3 are of course satisfied.

The set of maps $\mathcal{H}$ is partially ordered by pointwise inclusion of maps. Thus, $h \leq h'$ if $h(e, q) \subseteq h'(e, q)$ for all $e \in \mathcal{P}(Q)$ and $q \in Q$. The following lemma states that this set of maps is closed under arbitrary unions.

Lemma 5.6 *If $\mathcal{H}'$ is a set of maps such that $\mathcal{H}' \subseteq \mathcal{H}$, then $\cup\mathcal{H}' \in \mathcal{H}$.*

Proof. The proof is similar to the one of Proposition 5.6. □

[8] We will see that e is in fact a condensed representation of such state estimates, consisting in the states reached directly after the last observable events.

Therefore, the set of maps $\mathcal{H}$ has a supremum that we denote by $h^\dagger = \cup\mathcal{H}$. Also, from Definition 5.3, it follows that $\mathcal{K} = \{h(\{q_0\}, q_0) : h \in \mathcal{H},\ h(\{q_0\}, q_0) \neq \emptyset\}$ and then that $K^\dagger = h^\dagger(\{q_0\}, q_0)$.

Now let $[w]_a^\dagger = [w]_a \cap K^\dagger \cap \Sigma^*\Sigma_a$ and define:

$$\begin{array}{rccl} \zeta : & K^\dagger & \to & \mathcal{P}(Q) \times Q \\ & w & \mapsto & \begin{cases} (\{q_0\}, \delta(w, q_0)) \text{ if } w \in \Sigma_{ua}^* \\ (\delta([w]_a^\dagger, q_0), \delta(w, q_0)) \text{ otherwise} \end{cases} \end{array}$$

When a word $w \in K^\dagger$ is generated by G, then the function $\zeta(w) = (e, q)$ gives the knowledge of both parts, e for the attacker and q for the control, regarding the states that G may have reached. Since the control is under full observation, the controller knows exactly which state have been reached. The attacker is partially observing the events of w, and then cannot know more that an estimated set of states that the system may have reached, as for the determinization procedure of Definition 4.1. For technical reasons, we do not consider state estimates as for the determinization, but a condensed version of this state estimates which consists in the set of states the system may have reach directly after the last observable event of Σ_a. The usual state estimates can be recovered from condensed state estimates by computing the states reachable with events of Σ_{ua}.

Based on this construction, we show that the supremal controlled language after w only depends on the configuration $\zeta(w) \in \mathcal{P}(Q) \times Q$. Formally, we show that the following diagram is commutative:

$$\begin{tikzcd} K^\dagger \arrow[r, "r"] \arrow[d, "\zeta"'] & r(K^\dagger) \\ \mathcal{P}(Q) \times Q \arrow[ur, "h^\dagger"'] & \end{tikzcd}$$

where r denotes the residuation function on $K^\dagger$, i.e. the map $r : w \mapsto w^{-1}K^\dagger$. If $r = h^\dagger \circ \zeta$, then r being surjective, so is $h^\dagger$. This entails that $|r(K^\dagger)| \leq |\mathcal{P}(Q) \times Q|$ and then the regularity of $K^\dagger$ as there is only a finite number of residual languages. This also suggests the algorithm for its effective construction, that will be presented in the next sections.

Theorem 5.3 *If $w \in K^\dagger$ and $\zeta(w) = (e, q)$, then $w^{-1}K^\dagger = h^\dagger(e, q)$. So the diagram above is commutative:*

$$r = h^\dagger \circ \zeta$$

The proof of Theorem 5.3 relies on two more definitions and lemmas.

Definition 5.4

$$\begin{array}{rrcl} \textit{Let } \widetilde{h}: & \mathcal{P}(Q) \times Q & \rightarrow & \mathcal{P}(\Sigma^*) \\ & (e,q) & \mapsto & \cup\{w^{-1}K^\dagger : w \in K^\dagger \wedge \zeta(w) = (e,q)\} \end{array}$$

Lemma 5.7 $\widetilde{h} \in \mathcal{H}$

Proof. Let $(e,q) \in \mathcal{P}(Q) \times Q$. If $\widetilde{h}(e,q) = \emptyset$, then properties (1) and (2) are satisfied. Suppose that $\widetilde{h}(e,q) \neq \emptyset$. Then, $\widetilde{h}(e,q)$ is a union of prefix-closed and controllable sublanguages of $l_q(M)$ which implies property (1). For item (2), let $\phi \in \varPhi$ and $u \in \widetilde{h}(e,q)$. Then $wu \in K^\dagger$ for some $w \in K^\dagger$ such that $\zeta(w) = (e,q)$. There exists $v \in K^\dagger$ such that $v \sim_a wu$ and $\delta(v,q_0) \notin F(\phi)$. If $w \in \Sigma_{ua}^*$ then $e = \{q_0\}$, $v \sim_a u$ and $\delta(v,q_0) \notin F(\phi)$. Otherwise, we can write $v = w'u'$ with $w' \in [w]_a^\dagger$ and $u' \sim_a u$. If we let $q' = \delta(w',q_0)$ then $q' \in e$, $u' \in \widetilde{h}(q',e)$ and $\delta(u',q') = \delta(w'u',q_0) \notin F(\phi)$ which shows property (2). Therefore $\widetilde{h} \in \mathcal{H}$. □

Definition 5.5

$$\begin{array}{rrcl} \textit{Let } [\![\cdot]\!]: & \mathcal{H} & \rightarrow & \mathcal{P}(\Sigma^*) \\ & h & \mapsto & \cup\{w\ h(e,q) : w \in K^\dagger \wedge \zeta(w) = (e,q)\} \end{array}$$

Lemma 5.8 *For all $h \in \mathcal{H}$, $[\![h]\!] \in \mathcal{K}$.*

Proof. Let $h \in \mathcal{H}$. We note first that $K^\dagger \subseteq [\![h]\!]$. It is clear that $[\![h]\!]$ is a prefix-closed and controllable sublanguage of L_0. Let $\phi \in \varPhi$ and $u \in [\![h]\!]$. Then $u = wv$ with $w \in K^\dagger$, $\zeta(w) = (e,q)$ and $v \in h(e,q)$. Because h satisfies property (2), there must exist $q' \in e$ and $u' \in h(e,q')$ such that $u' \sim_a u$ and $\delta(u',q') \notin F(\phi)$. Also, there must exist $w' \in [w]_a^\dagger$ such that $\delta(w',q_0) = q'$ by definition of ζ. So $w'u' \sim_a wu$ and $\delta(w'u',q_0) = \delta(u',q') \notin F(\phi)$. Finally $[\![h]\!] \in \mathcal{K}$ and then $[\![h]\!] \subseteq K^\dagger$. □

We can now come back to the proof of Theorem 5.3.

Proof.(of Theorem 5.3) Let $w \in K^\dagger$ such that $\zeta(w) = (e,q)$. Clearly, $w^{-1}K^\dagger \subseteq \widetilde{h}(e,q)$ by definition of $\widetilde{h}$. According to lemma 5.7, $\widetilde{h}(e,q) \subseteq h^\dagger(e,q)$ since $\widetilde{h} \in \mathcal{H}$, so $w^{-1}K^\dagger \subseteq h^\dagger(e,q)$. Also, from definition 5.5, $[\![h^\dagger]\!] \subseteq K^\dagger$. Since $h^\dagger \in \mathcal{H}$ from Lemma 5.8, $h^\dagger(e,q) \subseteq w^{-1}K^\dagger$. Therefore, $w^{-1}K^\dagger = h^\dagger(e,q)$. □

Corollary 5.2 *The language $K^\dagger$ is regular.*

The corollary 5.2 shows that under the assumption $\Sigma_a \subseteq \Sigma_o$, the supremal controlled language enforcing the opacity on M is always regular.

An Informal Presentation of the Constructions

In this section, we give an informal presentation of the proposed algorithm for solving Problem 5.1.

We have seen with Theorem 5.3 that the residual languages $w^{-1}K^\dagger$ only depend on the configurations $\zeta(w) = (e, q)$ where, intuitively, e represents the knowledge of the attacker about the current state of G and q the one of the controller[9]. The first key idea of the algorithm is: instead of computing $K^\dagger$, we only need to compute the set of events $h^\dagger(e, q) \cap \Sigma$ as it is sufficient to recover $K^\dagger$. Indeed, let $w \in K^\dagger$ such that $\zeta(w) = (e, q)$ and $\sigma \in \Sigma$. Then, according to Theorem 5.3, $\sigma \in w^{-1}K^\dagger \iff \sigma \in h^\dagger(e, q) \cap \Sigma$. We can iterate the process for $(w\sigma)^{-1}K^\dagger$ and construct the language $K^\dagger$ by induction.

The second key idea is: the set of events $h^\dagger(e, q) \cap \Sigma$ actually induce a restriction of the transition function of G, i.e. we have $h^\dagger(e, q) \cap \Sigma \subseteq \delta(\cdot, q)^{-1}(Q)$, and this restriction only depends on the estimate e. In other words, if the attacker knows the state estimate $e \in \mathcal{P}(Q)$, then the language $h^\dagger(e, q), q \in Q$ induces the transition function $\Sigma \times Q \to Q$, $(\sigma, q) \mapsto \delta(\sigma, q)$ if $\sigma \in h^\dagger(e, q) \cap \Sigma$ or undefined otherwise. Then, assume that we know the set of events $h^\dagger(e, q) \cap \Sigma$ for each (reachable) configuration (e, q), we can define the partial function

$$\begin{array}{rcccl} d^\dagger : & \mathcal{P}(Q) & \to & (\Sigma \times Q & \to \quad Q) \\ & e & \mapsto & \sigma,\ q & \mapsto \quad \delta(\sigma, q) \text{ if } \sigma \in h^\dagger(e, q) \cap \Sigma \end{array}$$

Then, given such a partial function $d^\dagger$, we can construct an LTS $A^\dagger$, such that $\mathcal{L}(A^\dagger) = K^\dagger$. Indeed, let $A^\dagger = (\Sigma, \mathcal{P}(Q) \times Q, \delta^\dagger, (\{q_0\}, q_0))$ where

$$\begin{array}{rlcl} \delta^\dagger : & \Sigma \times (\mathcal{P}(Q) \times Q) & \to & (\mathcal{P}(Q) \times Q) \\ & \sigma,\ (e, q) & \mapsto & (e', d^\dagger(e)(\sigma, q)) \text{ where } e' = \begin{cases} e \text{ if } \sigma \notin \Sigma_a \\ d^\dagger(e)(\Sigma_{ua}^*\sigma, e) \text{ otherwise} \end{cases} \end{array}$$

To prove that $\mathcal{L}(A^\dagger) = K^\dagger$, let $(\{q_0\}, q_0) \xrightarrow{w} (e, q)$ be a run of $A^\dagger$. By looking at the definition of $d^\dagger$, it is clear that

$$\delta^\dagger(\cdot, (e, q))^{-1}(\mathcal{P}(Q) \times Q) = h^\dagger(e, q) \cap \Sigma = w^{-1}K^\dagger \cap \Sigma$$

By induction, we can prove that this implies

$$\mathcal{L}(A^\dagger, (e, q), \mathcal{P}(Q) \times Q) = w^{-1}K^\dagger$$

[9] Recall that the controller observes all Σ. In that case the state estimates are the states of G.

which, being true for every $(e,q) \in \mathcal{P}(Q) \times Q$, corresponds to the fact that $\mathcal{L}(A^\dagger) = K^\dagger$.

Then, the problem of computing $K^\dagger$ is replaced by the problem of computing the function $d^\dagger$. To compute $d^\dagger$, we consider the set of functions $d : \mathcal{P}(Q) \to (\Sigma \times Q \to Q)$ such that each function $d(e)$ represents a restriction of δ candidate to be $d^\dagger$. Given a restriction d defined in such a way, we can compute the LTS A_d, defined like $A^\dagger$ but for the function d, generating the language $L_d = \mathcal{L}(A_d) \subseteq L_0$ induced by d. From this language L_d, we will compute the set of controllable transitions $(e,q) \xrightarrow{\sigma} (e',q')$, where $\sigma \in \Sigma_c$, that must be disabled to avoid secret information to be disclosed. This transition should then be removed from $d(e)$.

Based on this process, we define an operator α removing from a function d the last controllable transition that must be disabled to avoid violating the opacity property with respect to the states $F(\phi), \phi \in \Phi$. Starting from the unrestricted function $d_0 : (e,\sigma,q) \mapsto \delta(\sigma,q)$, in which case $L_{d_0} = L_0$, we iterate the operator α until the fixpoint $D = gfp(d \to d_0 \cap \alpha(d))$ is reached. We know that such fixpoint will be reached as there is only finitely many functions d. Then, we prove that $D = d^\dagger$, or more precisely that the obtained language L_D is the supremal controlled language enforcing opacity, i.e. $L_D = \mathcal{L}(A_D) = K^\dagger$ which solves item (3) of Problem 5.1.

The Technical Development

We now present the algorithm informally described above. Throughout the section, $d : \mathcal{P}(Q) \to (\Sigma \times Q \to Q)$ denotes a partial function such that $d(e)(\sigma,q)$ is either equal to $\delta(\sigma,q)$ or undefined. The unrestricted function is given by $d_0 : e \mapsto \delta$. We first present the different definitions that are needed to compute the fixpoint D suggested above.

Definition 5.6 *Given a function $d : \mathcal{P}(Q) \to (\Sigma \times Q \to Q)$, we define inductively, for each $e \subseteq Q$, $d(e)(\epsilon,q) = q$ and $d(e)(\sigma w,q) = d(e)(w, d(e)(\sigma,q))$ for $\sigma \in \Sigma_{ua}$ and $w \neq \epsilon$. The language induced by d is given by the LTS $A_d = (\Sigma, \mathcal{P}(Q) \times Q, \delta_d, (\{q_0\}, q_0))$, where*

$$\begin{array}{rccl} \delta_d : & \Sigma \times (\mathcal{P}(Q) \times Q) & \to & (\mathcal{P}(Q) \times Q) \\ & \sigma, (e,q) & \mapsto & (e', d(e)(\sigma,q)) \text{ where } e' = \begin{cases} e \text{ if } \sigma \notin \Sigma_a \\ d(e)(\Sigma_{ua}^*\sigma, e) \text{ otherwise} \end{cases} \end{array}$$

The LTS A_d is deterministic and the transition function δ_d is extended to words in the usual way. Finally let $L_d = \mathcal{L}(A_d)$.

Remark 5.6 *$L_d \subseteq L_0$ by definition of δ_d.*

Remark 5.7 *If $d' \subseteq d$ in the sense that $d(e)(\sigma,q)$ is defined and equal to $d'(e)(\sigma,q)$ whenever the latter is defined, then for any $w \in \Sigma^*$, $\delta_{d'}(w,(e,q)) = (e',q')$ entails that*

$\delta_d(w,(e,q)) = (e'', q')$ for some $e'' \supseteq e'$. Therefore,

$$d' \subseteq d \implies L_{d'} \subseteq L_d$$

The following three lemmas show that if a sequence of transitions $(e_i, q_i) \xrightarrow{\sigma_i} (e_{i+1}, q_{i+1})$, $i = 1 \dots n$, is generated from $(e_1, q_1) = (\tilde{e}, \tilde{q})$ using the transition map δ_d, then for each i, $d(e_{i+1})(\Sigma_{ua}^*, e_{i+1})$ is the best estimate of the state $q_i = \delta(\sigma_1 \dots \sigma_i, \tilde{q})$ that the attacker can obtain from $\pi_a(\sigma_1 \dots \sigma_i)$ and the knowledge that $\tilde{q} \in d(\tilde{e})(\Sigma_{ua}^*, \tilde{e})$.

Lemma 5.9 *Let $w, w' \in L_d$ such that $\delta_d(w,(\tilde{e},\tilde{q})) = (e,q)$ and $\delta_d(w',(\tilde{e},\tilde{q})) = (e',q')$. Then $\pi_a(w) = \pi_a(w') \implies e = e'$.*

Proof. We use an induction on the length of $\pi_a(w)$. If this length is zero, i.e. $w, w' \in \Sigma_{ua}^*$, then $e = \tilde{e}$ and $e' = \tilde{e}$ by Definition 5.6, hence $e = e'$. Assume now that the lemma holds when $\pi_a(w)$ has length n, and consider $w, w' \in \Sigma^*$ with $\pi_a(w) = \pi_a(w') \in \Sigma_a^{n+1}$. Let $w = w_1 \sigma w_2$ and $w' = w_1' \sigma w_2'$ such that $\pi_a(w_1) = \pi_a(w_1') \in \Sigma_a^n$ and $\sigma \in \Sigma_a$, hence $w_2, w_2' \in \Sigma_{ua}^*$. By the induction hypothesis, if we let $\delta_d(w_1,(\tilde{e},\tilde{q})) = (e_1,q_1)$ and $\delta_d(w_1',(\tilde{e},\tilde{q})) = (e_1',q_1')$, then $e_1 = e_1'$. By Definition 5.6, if we let $\delta_d(w_1\sigma,(\tilde{e},\tilde{q})) = (e_2,q_2)$ and $\delta_d(w_1'\sigma,(\tilde{e},\tilde{q})) = (e_2',q_2')$ then $e_2 = d(e_1)(\Sigma_{ua}^*\sigma, e_1)$ and $e_2' = d(e_1')(\Sigma_{ua}^*\sigma, e_1')$, hence $e_2 = e_2'$. Now $(e,q) = \delta_d(w_2,(e_2,q_2))$ and $(e',q') = \delta_d(w_2',(e_2',q_2'))$. As $w_2, w_2' \in \Sigma_{ua}^*$, by Definition 5.6, $e = e_2$ and $e' = e_2'$, hence $e = e'$. □

Lemma 5.10 *If $\tilde{q} \in d(\tilde{e})(\Sigma_{ua}^*, \tilde{e})$, then $\delta_d(w,(\tilde{e},\tilde{q})) = (e,q)$ implies $q \in d(e)(\Sigma_{ua}^*, e)$.*

Proof. If $w = \epsilon$, then $\delta_d(w,(\tilde{e},\tilde{q})) = (\tilde{e},\tilde{q})$ and the property to show coincides with the hypothesis about $(\tilde{e},\tilde{q})$. If $w = w'\sigma$ with $\sigma \in \Sigma$, let $\delta_d(w',(\tilde{e},\tilde{q})) = (e',q')$. One may assume by induction on words that $q' \in d(e')(\Sigma_{ua}^*, e')$, thus $q' = d(e')(w'',q'')$ for some $q'' \in e'$ and $w'' \in \Sigma_{ua}^*$. Then $q = d(e')(\sigma,q') = d(e')(\sigma, d(e')(w'',q'')) = d(e')(w''\sigma, q'') \in d(e')(\Sigma_{ua}^*\sigma, e')$. One proceeds by cases. Suppose that $\sigma \notin \Sigma_a$, then $\delta_d(\sigma,(q',e')) = (e,q)$ entails $e = e'$ and the desired result follows from $q \in d(e')(\Sigma_{ua}^*, e')$. Suppose that $\sigma \in \Sigma_a$, then $\delta_d(\sigma,(q',e')) = (e,q)$ entails $e = d(e')(\Sigma_{ua}^*\sigma, e')$ and therefore $q \in e$. The desired result follows from $e \subseteq d(e)(\Sigma_{ua}^*, e)$. □

Lemma 5.11 *If $\delta_d(w,(\{q_0\},q_0)) = (e,q)$ and $q' \in d(e)(\Sigma_{ua}^*, e)$, then there exists $w' \in L_d$ such that $\pi_a(w) = \pi_a(w')$ and $\delta_d(w',(\{q_0\},q_0)) = (e,q')$.*

Proof. We use an induction on the length of $\pi_a(w)$. Suppose that $|\pi_a(w)| = 0$. By Definition 5.6, $e = (\{q_0\})$, hence $q' = d(\{q_0\})(w',q_0)$ for some $w' \in \Sigma_{ua}^*$, and $\pi_a(w) = \pi_a(w')$. By Definition 5.6, $\delta_d(w',(\{q_0\},q_0)) = (\{q_0\}, d(\{q_0\})(w',q_0)) = (e,q')$ as desired. Assume now that the proposition holds when $\pi_a(w)$ has length n, and consider now $w \in \Sigma^*$ with $\pi_a(w) \in \Sigma_a^{n+1}$. Let $w = w_1\sigma w_2$ such that $\pi_a(w_1) \in \Sigma_a^n$ and $\sigma \in \Sigma_a$, hence $w_2 \in \Sigma_{ua}^*$. Let $\delta_d(w_1, (\{q_0\}, q_0)) = (e_1, q_1)$. By Definition 5.6, $e = d(e_1)(\Sigma_{ua}^*\sigma, e_1)$. As $q' \in d(e)(\Sigma_{ua}^*, e)$, $q' = d(e)(w_2', q_2')$ for some $q_2' \in e$ and $w_2' \in \Sigma_{ua}^*$. As $e = d(e_1)(\Sigma_{ua}^*\sigma, e_1)$, $q_2' = d(e_1)(w_2''\sigma, q_1')$ for some $q_1' \in e_1$ and $w_2'' \in \Sigma_{ua}^*$. As $\pi_a(w_1)$ has length n, the induction hypothesis applies to $\delta_d(w_1, (\{q_0\}, q_0)) = (e_1, q_1)$ and to $q_1' \in e_1 \subseteq d(e_1)(\Sigma_{ua}^*, e_1)$. Therefore, $\delta_d(w_1', (\{q_0\}, q_0)) = (e_1, q_1')$ for some w_1' such that $\pi_a(w_1) = \pi_a(w_1')$. Now $\delta_d(w_1'w_2''\sigma, (\{q_0\}, q_0)) = (d(e_1)(e_1, \Sigma_{ua}^*\sigma), q_2') = (e, q_2')$, hence $\delta_d(w_1'w_2''\sigma w_2', (\{q_0\}, q_0)) = (e, q')$, establishing the lemma. $\square$

We will now investigate which words in L_d actually disclose to the attacker the secret predicates $\phi \in \Phi$ defined by the accepting states $F(\phi) \subseteq Q$. We show how one can remedy these security failures. First, let us give a definition.

Definition 5.7 *Given a partial function $d : \mathcal{P}(Q) \to (\Sigma \times Q \to Q)$, let $\mathcal{LE}(d) = \{e \subseteq Q : e \neq \emptyset \wedge \exists \phi \in \Phi, d(e)(\Sigma_{ua}^*, e) \subseteq F(\phi)\}$ be the associated set of* loosing estimates, *and for any $(e, q) \in \mathcal{P}(Q) \times Q$ such that $q \in d(e)(\Sigma_{ua}^*, e)$, let*

$$\mathcal{LT}(d, (e, q)) = \{w \in \Sigma^* : \delta_d(w, (e, q)) \in \mathcal{LE}(d) \times Q\}$$

be the set of loosing words *w.r.t. the state q, the attacker state estimate e and the restriction d.*

The subset of words in L_d that disclose the secret may now be recognized by the automaton A_d (see Definition 5.6) equipped with the set of accepting states $\mathcal{LE}(d) \times Q$, as stated in the following proposition.

Proposition 5.12 *For any $w \in L_d$, $[w]_a \cap L_d \subseteq L(\phi)$ if and only if $\delta_d(w, (\{q_0\}, q_0)) \in \mathcal{LE}(d) \times Q$.*

Proof. Let $w \in L_d$ such that $[w]_a \cap L_d \subseteq L(\phi)$. Let $\delta_d(w, (\{q_0\}, q_0)) = (e, q)$, and let $q' \in d(e)(\Sigma_{ua}^*, e)$. By Lemma 5.11, $(e, q') = \delta_d(w', (\{q_0\}, q_0))$ for some $w' \in [w]_a \cap L_d$. As $w' \in [w]_a$ and $w' \in L_d$, we have $w' \in [w]_a \cap L_d \subseteq L(\phi)$. As $\delta(w', q_0) = q'$, it follows that $q' \in F(\phi)$. Therefore, $e \in \mathcal{LE}(d)$. To show the converse implication, let $\delta_d(w, (\{q_0\}, q_0)) = (e, q) \in \mathcal{LE}(d) \times Q$. Hence $d(e)(\Sigma_{ua}^*, e) \subseteq F(\phi)$ according to Definition 5.7. By Lemma 5.9, for any $w' \in [w]_a \cap L_d$, if we let $q' = \delta(w', q_0)$ then $\delta_d(w', (\{q_0\}, q_0)) = (e, q')$. By Lemma 5.10, $q' \in d(e)(\Sigma_{ua}^*, e) \subseteq F(\phi)$. Therefore, $[w]_a \cap L_d \subseteq L(\phi)$. $\square$

Proposition 5.13 *If $(e, q) = \delta_d(w, (\{q_0\}, q_0))$, then*

$$\mathcal{LT}(d, (e, q)) = \{w' \in \Sigma^* : ww' \in L_d \wedge [ww']_a \cap L_d \subseteq L(\phi)\}$$

Proof. Let $w' \in \mathcal{LT}(d,(e,q))$, then by definition, $\delta_d(ww',(\{q_0\},q_0)) = (e',q')$ with $e' \in \mathcal{LE}(d)$. By Proposition 5.12, $[ww']_a \cap L_d \subseteq L(\phi)$. To prove the converse inclusion relation, consider $w' \in \Sigma^*$ such that $ww' \in L_d$ and $[ww']_a \cap L_d \subseteq L(\phi)$. Let $\delta_d(ww',(\{q_0\},q_0)) = (e',q')$. By Proposition 5.12, $\delta_d(ww',(\{q_0\},q_0)) \in \mathcal{LE}(d) \times Q$, hence $e' \in \mathcal{LE}(d)$. Now $\delta_d(ww',(\{q_0\},q_0)) = \delta_d(w',(e,q))$, hence $w' \in \mathcal{LT}(d,(e,q))$. □

Proposition 5.13 proves that, if $(e,q) = \delta_d(w,(\{q_0\},q_0))$ for some trace $w \in L_0$ and some partial function $d : \mathcal{P}(Q) \to \Sigma \times Q \to Q$, then for any $w' \in \mathcal{LT}(d,(e,q))$, if an attacker gets the state estimate e immediately after the trace w has been executed in $G \parallel A_d$, then the attacker can infer from the projection $\pi_a(w')$ of the subsequent trace w' executed in $G \parallel A_d$ that ww' is in the secret set $L(\phi)$[10]. More generally, even though the configuration (e,q) may not be reachable in A_d, if $w' \in \mathcal{LT}(d,(e,q))$ and $(e,q) = \delta_{d'}(w,(\{q_0\},q_0))$ for some $d' \subseteq d$ and $w \in \Sigma^*$, then $w' \in \mathcal{LT}(d',(e,q))$.

Based on Proposition 5.13, we immediately have the following.

Corollary 5.3 *If $\mathcal{LT}(d,(e,q)) = \emptyset$ for every configuration $(e,q) = \delta_d(w,(\{q_0\},q_0))$ reached in A_d, then $Disclose(L_d,\pi_a)(\Phi) = \emptyset$, i.e. L_d is opaque.*

We have now in hands all elements needed to compute $D : \mathcal{P}(Q) \to (\Sigma \times Q \to Q)$ such that $Disclose(L_D,\pi_a)(\Phi) = \emptyset$ and $L_D = K^\dagger$ is the largest controllable sublanguage of L_0 with this property.

Next, we define the map α removing from a function d the transitions inducing a secret predicate to be disclosed in A_d. This definition of α present some similarities with the construction of the operator CN in Section 5.1.

Definition 5.8 *Given $d : \mathcal{P}(Q) \to (\Sigma \times Q \to Q)$, let $\alpha(d) \subseteq d$ be the partial function such that*

- *$\alpha(d)(e)(\sigma,q)$ is undefined if $\sigma \in \Sigma_c$ and $\mathcal{LT}(d,(e',q')) \cap \Sigma_{uc}^* \neq \emptyset$ for $(e',q') = \delta_d(\sigma,(e,q))$,*
- *$\alpha(d)(e)(\sigma,q) = d(e)(\sigma,q)$ otherwise.*

It is important to note that deciding whether the set $\mathcal{LT}(d,(e',q')) \cap \Sigma_{uc}^*$ is empty is a reachability problem that can be checked on the finite automaton generated by the partial transition map δ_d from the initial state (e',q').

Now, let the partial function D be defined by:

$$D = gfp(d \to d_0 \cap \alpha(d))$$

[10]This also holds for the original system M since $\mathcal{L}(G) = \mathcal{L}(M)$

The partial function D is well defined since $\alpha(d) \subseteq d$ for all d and there exist only finitely many partial functions $d : \mathcal{P}(Q) \to (\Sigma \times Q \to Q)$. Note that $\mathcal{L}(d_0) = L_0$. The following proposition shows the controllability of the languages obtained by iterating α from d_0.

Proposition 5.14 *If L_d is controllable, then $L_{\alpha(d)}$ is also controllable.*

Proof. Let $w \in L_{\alpha(d)}$ and $\sigma \in \Sigma_{uc}$ such that $w\sigma \in L_0$. Since $L_{\alpha(d)} \subseteq L_d$, $w \in L_d$. As L_d is controllable, then $w\sigma \in L_d$. If $w\sigma \in L_d \setminus L_{\alpha(d)}$, then we must have $\sigma \in \Sigma_c$ according to Definition 5.8, which is not possible. □

Let $d_i = \alpha^i(d_0)$, for $i \in \mathbb{N}$. As the language $L_{d_0} = L_0$ is controllable, the proposition 5.14 implies by induction that all the $L_{d_i}, i \in \mathbb{N}$ are controllable. This implies that:

Corollary 5.4 *L_D is controllable*

The following two propositions show that the language L_D is the largest controllable sublanguage of L_0 such that $K^\dagger$ is opaque.

Proposition 5.15 *L_D is opaque.*

Proof. Assume for contradiction that L_D is not opaque. Then there exists $w \in L_D$ such that $[w]_a \cap L_D \subseteq L(\phi)$. By Proposition 5.12, $w \in \mathcal{LT}(D, (\{q_0\}, q_0))$. We claim that $w \in \Sigma_{uc}^*$. In order to establish this property, assume for contradiction that $w = w_1\sigma w_2$ with $\sigma \in \Sigma_c$ and $w_2 \in \Sigma_{uc}^*$. Let $\delta_D(w_1, (\{q_0\}, q_0)) = (q_1, e_1)$ and $\delta_D(\sigma, (e_1, q_1)) = (e_2, q_2)$, then by Proposition 5.13, $w_2 \in \mathcal{LT}(D, (e_2, q_2))$. As $w_2 \in \Sigma_{uc}^*$, by Definition 5.8, $\alpha(D)(e_1)(\sigma, q_1)$ is undefined. As $D(e_1)(\sigma, q_1) = q_2$, it is impossible that $D = \alpha(D)$. It follows from this contradiction that $w \in \Sigma_{uc}^*$. Recalling that $L_D \subseteq L_0$, we observe now that necessarily $w = \epsilon$, because $w \in \Sigma_{uc}^*$ and $\delta(\sigma, q_0)$ is undefined for all $\sigma \in \Sigma_{uc}$. Now $[\epsilon]_a \cap L_D \not\subseteq L(\phi)$ since it has been assumed that $q_0 \notin F(\phi)$, hence it is impossible that $[w]_a \cap L_D \subseteq L(\phi)$. It follows from this second contradiction that L_D is opaque. □

Proposition 5.16 *Let K be any controlled sublanguage of L_0 such that K is opaque. Then $K \subseteq L_{d_i}$ for all $i \in \mathbb{N}$.*

Proof. In order to establish the proposition, we assume that $K \not\subseteq L_{d_i}$ for some i and we search for a contradiction. As $L_{d_i} \supseteq L_{d_{i+1}}$ for all $i \in \mathbb{N}$, we can moreover assume that i is the least integer such that $K \not\subseteq L_{d_i}$. As $L_0 = L_{d_0}$, we have $i \neq 0$. Let w be a minimal word w.r.t. the prefix order in $K \setminus L_{d_i}$. As L_{d_i} is prefix-closed, $w \neq \epsilon$. Let $w = w'\sigma$ with $\sigma \in \Sigma$. As w has no strict prefix in $K \setminus L_{d_i}$, necessarily $w' \in K \cap L_{d_i}$. Thus, $w' \in L_{d_i}$, $w'\sigma \notin L_{d_i}$, and $w'\sigma \in L_{d_{i-1}}$ since $w'\sigma = w \in K \subseteq L_{d_{i-1}}$. By construction of the map $d_i = \alpha(d_{i-1})$, $\sigma \in \Sigma_c$ and $\mathcal{LT}(d_{i-1}, (e, q)) \cap \Sigma_{uc}^* \neq \emptyset$ for $(e, q) = \delta_{d_{i-1}}(w'\sigma, (\{q_0\}, q_0))$. By

Proposition 5.13, there exists $w'' \in \Sigma_{uc}^*$ such that $w'\sigma w'' \in L_{d_{i-1}}$ and $[w'\sigma w'']_a \cap L_{d_{i-1}}$ is included in $L(\phi)$. Now $w = w'\sigma$ is in K, $w'\sigma w''$ is in L_0 because $L_{d_{i-1}} \subseteq L_0$, and $w'' \in \Sigma_{uc}^*$. As K is a controllable sublanguage of L_0, it follows that $w'\sigma w''$ is in K. As $[w'\sigma w'']_a \cap L_{d_{i-1}} \subseteq L(\phi)$ and $K \subseteq L_{d_{i-1}}$ by assumption on i, $[w'\sigma w'']_a \cap K$ is included in $L(\phi)$. This contradicts the hypothesis that K is opaque. Therefore, the proposition has been established. □

Theorem 5.4 *L_D is the supremal controlled sublanguage of L_0 s.t. L_D is opaque, i.e.*

$$L_D = K^\dagger$$

Proof. Follows directly Propositions 5.15 and 5.16. □

The Theorem 5.4 solves the Problem 5.1 as it shows the correctness of an algorithm to compute the supremal controlled language enforcing the opacity of Φ on G, and by consequence on M.

Illustration of the algorithm with an example

We new illustrate the algorithm above with a small example. For this, consider the LTS of Figure 5.3 that we give again in figure 5.7. We keep the same hypothesis, i.e. $\Sigma_a = \{a, b, X, Y\}$, $\Sigma_{ua} = \{c, u\}$, $\Sigma_c = \{c\}$ and $\Sigma_o = \Sigma$. The secret predicate is defined by the set of the sequences that reach the state labeled by 5 in the figure. It is already patent from

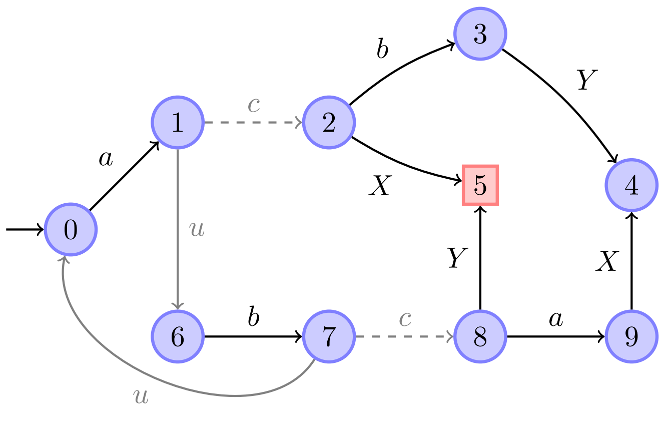


Figure 5.7: G

the figure 5.7, that letting the transitions labeled by c at state (1) and (7) will lead the secret to be disclosed. On this example, we will detail step by step the algorithm presented above to prove that the supremal opaque controlled language of $L_0 = \mathcal{L}(G)$ is $(aubu)^*$.

The first step consists in extracting from G the partial function d_0. For this, we use a property of the functions d that was implicit in the presentation of the algorithm: we only need to define $d(e)$ for the state that are reached in G by words of the form $w\sigma$ with $w \in \Sigma_{ua}^*$ and $\sigma \in \Sigma_a$. Indeed, as soon as an event $\sigma \in \Sigma_a$ occurs in G, the state estimate e of the attacker is updated to the new one given by $e' = d(e)(\sigma, d(e)(\Sigma_{ua}^*, e))$. Therefore, d_0 is given as follows:

$d_0(\{0\})(a, 0) = 1$

$d_0(\{1\})(c, 1) = 2,\ d_0(\{1\})(u, 1) = 6$
$d_0(\{1\})(X, 2) = 5,\ d_0(\{1\})(b, 2) = 3,\ d_0(\{1\})(b, 6) = 7$

$d_0(\{3\})(Y, 3) = 4$

$d_0(\{7\})(c, 7) = 8),\ d_0(\{7\})(u, 7) = 0)$
$d_0(\{7\})(a, 0) = 1),\ d_0(\{7\})(a, 8) = 9),\ d_0(\{7\})(Y, 8) = 5)$

$d_0(\{9\})(X, 9) = 4$

$d_0(\{3, 7\})(c, 7) = 8),\ d_0(\{3, 7\})(u, 7) = 0)$
$d_0(\{3, 7\})(a, 0) = 1),\ d_0(\{3, 7\})(a, 8) = 9),\ d_0(\{3, 7\})(Y, 3) = 4),\ d_0(\{3, 7\})(Y, 8) = 5)$

$d_0(\{1, 9\})(c, 1) = 2),\ d_0(\{1, 9\})(u, 1) = 6)$
$d_0(\{1, 9\})(b, 2) = 3),\ d_0(\{1, 9\})(b, 6) = 7),\ d_0(\{1, 9\})(X, 2) = 5),\ d_0(\{1, 9\})(X, 9) = 4)$

From d_0, we can now derive the transition function δ_{d_0} and the automaton A_{d_0} depicted in Figure 5.8. On the figure 5.8, we see that the state $(\{5\}, 5)$ is such that the secret is disclosed. Therefore, backtracking until the last controllable transition, we see that the transition $(\{1\}, 1) \xrightarrow{c} (\{1\}, 2)$ need to be removed in A_{d_0}. According to the definition of α, the function $d_1 = \alpha(d_0)$ is obtained by removing $d_0(\{1\})(c, 1) = 2$ from d_0. We obtain then the function d_1, given by:

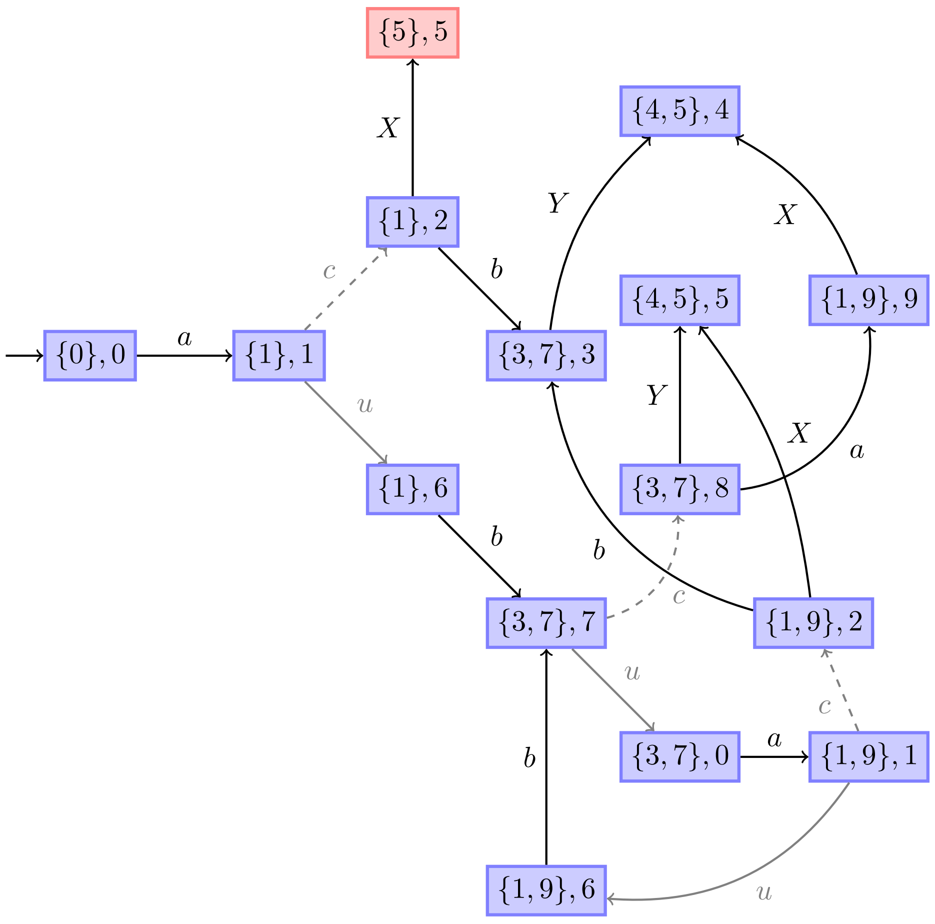


Figure 5.8: A_{d_0}

$d_1(\{0\})(a,0) = 1$

$d_1(\{1\})(u,1) = 6$
$d_1(\{1\})(b,6) = 7$

$d_1(\{3\})(Y,3) = 4$

$d_1(\{7\})(c,7) = 8),\ d_1(\{7\})(u,7) = 0)$
$d_1(\{7\})(a,0) = 1),\ d_1(\{7\})(a,8) = 9),\ d_1(\{7\})(Y,8) = 5)$

$d_1(\{9\})(X,9) = 4$

$d_1(\{3,7\})(c,7) = 8),\ d_1(\{3,7\})(u,7) = 0)$
$d_1(\{3,7\})(a,0) = 1),\ d_1(\{3,7\})(a,8) = 9),\ d_1(\{3,7\})(Y,3) = 4),\ d_1(\{3,7\})(Y,8) = 5)$

$d_1(\{1,9\})(c,1) = 2),\ d_1(\{1,9\})(u,1) = 6)$
$d_1(\{1,9\})(b,2) = 3),\ d_1(\{1,9\})(b,6) = 7),\ d_1(\{1,9\})(X,2) = 5),\ d_1(\{1,9\})(X,9) = 4)$

Note that removing $d_0(\{1\})(c,1) = 2$ from d_0 also implies that $d_0(\{1\})(X,2) = 5$ and $d_0(\{1\})(b,2) = 3$ have disappeared in d_1 as the configuration $(\{1\},2)$ is no more reachable. From d_1, we obtain the LTS A_{d_1} depicted in Figure 5.9.

Now, according to the LTS A_{d_1}, we need to remove the transition $d_1(\{7\})(c,7) = 8$ to

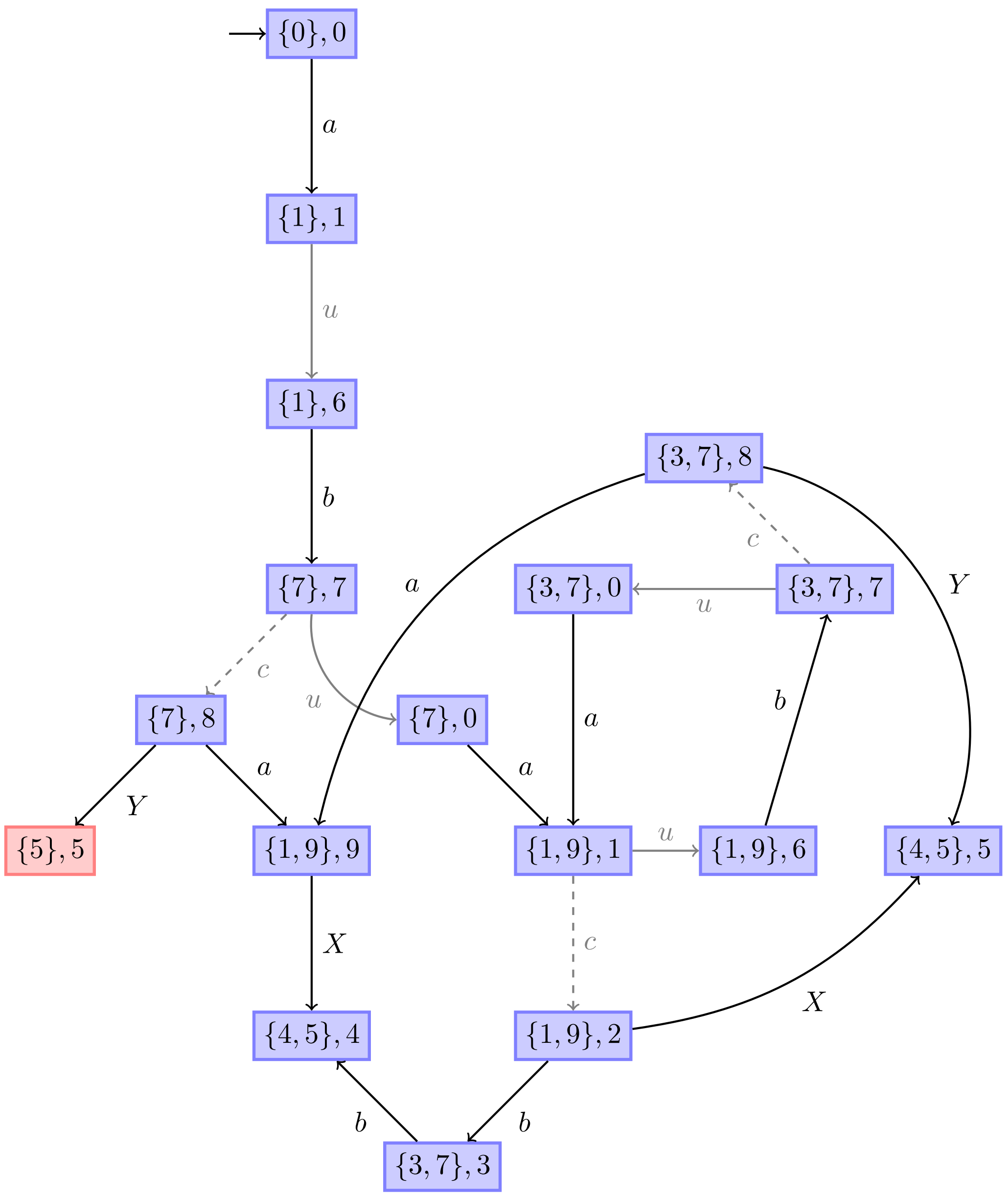


Figure 5.9: A_{d_1}

obtain $d_2 = \alpha(d_1)$. Thus, the function d_2 is obtained as follows:

$d_2(\{0\})(a, 0) = 1$

$d_2(\{1\})(u, 1) = 6$
$d_2(\{1\})(b, 6) = 7$

$d_2(\{3\})(Y, 3) = 4$

$d_2(\{7\})(u, 7) = 0)$
$d_2(\{7\})(a, 0) = 1)$

$d_2(\{9\})(X, 9) = 4$

$d_2(\{3, 7\})(c, 7) = 8),\ d_2(\{3, 7\})(u, 7) = 0)$
$d_2(\{3, 7\})(a, 0) = 1),\ d_2(\{3, 7\})(a, 8) = 9),\ d_2(\{3, 7\})(Y, 3) = 4),\ d_2(\{3, 7\})(Y, 8) = 5)$

$d_2(\{1, 9\})(c, 1) = 2),\ d_2(\{1, 9\})(u, 1) = 6)$
$d_2(\{1, 9\})(b, 2) = 3),\ d_2(\{1, 9\})(b, 6) = 7),\ d_2(\{1, 9\})(X, 2) = 5),\ d_2(\{1, 9\})(X, 9) = 4)$

The LTS A_{d_2} is depicted in Figure 5.10.

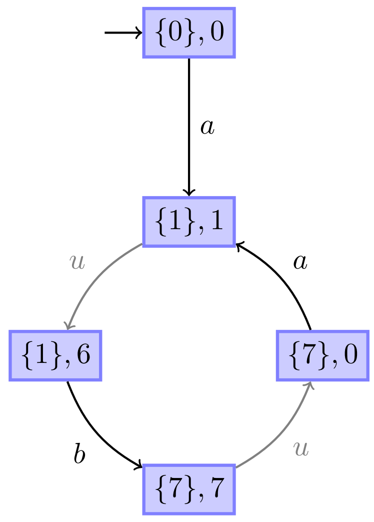

$$\begin{tikzcd}
\arrow[r] & \{0\},0 \arrow[d, "a"] & \\
& \{1\},1 \arrow[dl, bend right, "u"'] & \\
\{1\},6 \arrow[dr, bend right, "b"'] & & \{7\},0 \arrow[ul, bend right, "a"'] \\
& \{7\},7 \arrow[ur, bend right, "u"'] &
\end{tikzcd}$$

Figure 5.10: A_{d_2}

We note that there is no loosing estimate in A_{d_2}. Therefore, $\alpha(d_2) = d_2$ and then $D = d_2$. We obtain finally that $K^\dagger = L_{d_2} = \mathcal{L}(A_{d_2})$, i.e. $K^\dagger = (aubu)^*$.

5.4 Conclusion

We consider in this chapter how to enforce opacity on a system modeled by a finite LTS. For the case $\Sigma_o \subseteq \Sigma_a$, we proved solution to this problem based on the fixpoint iteration technique of Ramadge and Wonham. For the case $\Sigma_a \subseteq \Sigma_o$, we give the example of a system such that this fixpoint iteration does not terminate after a finite number of iterations. We develop then an new technique to provide a solution to the opacity control problem in that case.

In the literature, security problems usually arise in the context of infinite systems, including for instance unbounded Petri nets. We provided solution to compute counterexample to opacity in Chapter 4. We can imagine to apply this control technique on an abstraction G of M, i.e. such that $C \parallel G$ is opaque and then apply the controller C to M. Then there is no guaranty that $C \parallel M$ is opaque but can repeat the operation of computing an abstraction G' of $C \parallel M$ and search for sufficient conditions such that this procedure terminates, therefore increasing the confidence in the security of M.

6 Dynamic Projections to Enforce Opacity

In the previous chapter, we presented how to enforce the opacity on a system by restricting its behavior. In this chapter, we consider an alternative approach to enforce opacity on a system M by computing a projection that dynamically changes the observability of events in such a way that attackers cannot infer secret information from the observations. For this purpose, we will develop the notion of *dynamic projection* and *observer* and consider the architecture depicted in Figure 6.1. Such a dynamic projection, say π, is implemented by an observer O added between the system and the user, taking as input the words generated by M and sending the observed traces, i.e. the outcomes of π. Such an observer O is a classical labeled transition system augmented by a map giving at each states of O, the events that are observable next. The results presented in this chapter follow the publication of [CDM09a].

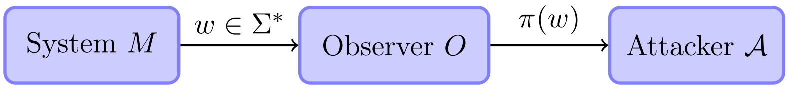


Figure 6.1: Architecture with an observer between M and the attacker

Whereas possible applications that we have in mind for the supervisory control approach were about automating the design of secure systems, possible applications that we imagine for this work about dynamic projections is the correction of existing systems where concerns of confidentiality are critical. Indeed, one of the advantages of dynamic projections is that their implementation is not intrusive, in the sense that it does not alter the behavior of M but only the observed traces. This implies that in the context of several and potentially malicious users $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n$, we can implement several observers $O_1, O_2, \ldots, O_n$, one for each user, and no observer for one attacker will influence what the other attackers can see and deduce from the system[1]. So, we can implement a dynamic projection that strongly restricts the observable behavior, and thus the provided service, for unprivileged users who shall not access critical information, and implement a more permissive one for authorized users. We can imagine for example a web service with different observability

[1]For example, this is not true for the controller synthesis approach. See for example [BBB$^+$07]

policies depending on the IP address of the users.

Given a finite LTS M and a finite set of state-based secret predicates Φ, we will develop in this chapter the following aspects. First, for static projections, we consider in 6.1 the optimization problem of computing a maximal set of observable events to ensure opacity and show that this problem is PSPACE-complete. Second, we formalize the notion of dynamic projection and the associated notion of observer. We show how to check opacity when the dynamic projection is given by a finite observer. Finally, we give an algorithm to compute the set of all dynamic projections enforcing opacity. We show that this problem can be reduced to a *safety 2-player game problem.* Intuitively, the first player will play the rôle of an observer and will decide which subset of event should remain observable after a given trace. The second player will play the rôle of both the system and the attacker and will decide what will be the next observable event among the ones Player 1 has last chosen to render observable. The goal of Player 2 is thus to make the system evolve in the states in which the attacker is sure that the secret is disclosed, whereas the goal of Player 1 is opposite (he has to choose the successive sequences of sets of observable events in such a way that the secret is never disclosed). It results from this reduction that the set of valid dynamic projections (the ones enforcing opacity) can be finitely represented by a game LTS. We also prove that this set can be computed in EXPTIME.

In the sequel, we will always consider opacity with respect to the set of state-based secret predicates Φ, so the term opacity will implicitly stand for Φ-opacity.

6.1 Maximum Cardinality Set for Static Projections

In this section, we investigate the problem of computing the largest set of observable events such that opacity is preserved. According to Proposition 3.3, if a secret is opaque w.r.t. to a set of observable events Σ_a, it will still be opaque w.r.t. any subset of Σ_a (the less you observe, the less accurate you are). It might be of interest to hide as few events as possible from the attacker while still preserving opacity of a secret.

Assume that Φ is opaque on M w.r.t. $\Sigma_a = \emptyset$. This is true whenever $Q \setminus F(\phi) \neq \emptyset$ for all $\phi \in \Phi$. This suggests an optimization problem which can be formulated as follows: What is the maximum cardinality of the sets of observable events $\Sigma_a \subseteq \Sigma$ such that the secret is opaque ? More precisely:

Problem 6.1 (Maximum Number of Observable Events)

- ***Inputs:*** *A finite LTS $M = (\Sigma, Q, \delta, Q_0)$ and a natural number $n \in \mathbb{N}$ s.t. $n \leq |\Sigma|$.*
- ***Problems:***

– ***(A)*** *Is there any $\Sigma_a \subseteq \Sigma$ with $|\Sigma_a| = n$, such that M is opaque w.r.t. Σ_a ?*
– ***(B)*** *If the answer to **(A)** is "yes", find the maximum n_{max} such that there exists $\Sigma_a \subseteq \Sigma$ with M opaque w.r.t. Σ_a and $|\Sigma_a| = n_{max}$.*

Note that we do not search directly for a supremal subset of observable events enforcing opacity as this supremal subset may not be unique. The following theorem states that this optimization problem is not harder than verifying opacity given a subset of observable events.

Theorem 6.1 *Problems 6.1 (A) and (B) are PSPACE-complete.*

Proof. PSPACE-easiness follows Theorem 4.2 as we can guess a set Σ_a with $|\Sigma_a| = n$ and check in PSPACE whether M is opaque w.r.t. Σ_a. Thus Problem 6.1 (A) is in NPSPACE and thus in PSPACE. PSPACE-hardness follows Proposition 4.4. Indeed, choosing $n = |\Sigma|$ and checking that M is opaque w.r.t. Σ which has been shown equivalent to the universality problem in 4.2.

To solve Problem 6.1 (B) it suffices to proceed to an exhaustive search (for every subset of Σ) and thus Problem 6.1 (B) is also in PSPACE. To check whether M is opaque w.r.t. Σ, it suffices to solve Problem 6.1 (B) and then check whether $n_{max} = |\Sigma|$. So this problem is also PSPACE-complete. $\square$

6.2 Opacity with Dynamic Projection

We now present the notion of dynamic projection and study the opacity verification problem for this category of observation maps.

We have assumed in the two previous chapters that the observability of events was given a priori as a projection (e.g. $\pi_{\Sigma \to \Sigma_a}$). We generalize this approach by studying the notion of *dynamic projections*. Such projections can be encoded by means of *observers* as introduced in [CT08] for the fault diagnosis problem. In this section, we introduce the notion of dynamic projection and observers and we investigate the opacity verification problem in this context.

A dynamic projection renders unobservable some events after a given observed trace (for example, some outputs of the system) and we search for dynamic projections enforcing the opacity on M. To illustrate the benefits of such projections, we consider the following example:

Example 6.1 *Consider the LTS M of Figure 6.2, with $\Sigma_a = \Sigma = \{a, b\}$. The secret is given by the states represented by the squares* 🟥 *i.e. $F(\phi) = \{q_2, q_5\}$. The system M is*

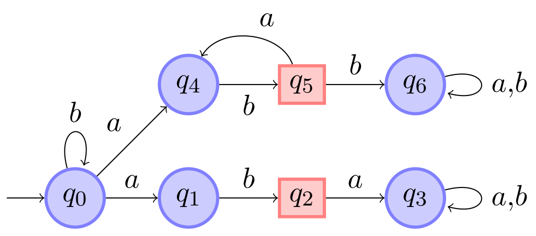


Figure 6.2: Illustration of Dynamic Projections

not opaque with respect to Σ *since an attacker can disclose the secret* ϕ *for every observed traces of* b^*ab.

Now, if either $\Sigma_a = \{a\}$ *or* $\Sigma_a = \{b\}$, *then the system becomes opaque. Thus if we have to define static sets of observable events, at least one event will have to be permanently unobservable. But, we can be less restrictive by using a dynamic projection that will render unobservable an event only when necessary. In this example, after observing* b^*, *the attacker still knows that the system is in the initial state. However, if a subsequent "a" follows, then the attacker should not be able to observe "b" as in this case the secret information is disclosed. We can then design a dynamic projection defined as follows: at the beginning, everything is observable; when an "a" occurs, the observer hides any subsequent occurrence of "b" and permits only the observation of "a". Once an "a" has been observed, the projection releases its policy by letting both "a" and "b" be observable.*

In this section, we define the notion of dynamic projection and present how the dynamic projections can be encoded by observers as introduced in [CT08].

Dynamic Projections and Observers

An (observation-based) dynamic projection is a function that will decide whether to let an event be observable according to the trace observed by the attacker. For this, we suppose that the alphabet of events is partitioned into a set Σ_a of events that can be observable (interactions, inputs/outputs) and internal events $\Sigma \setminus \Sigma_a$ that are always unobservable. The secret predicates being state based, the internal events are all playing a symmetrical rôle in the following. Then, to simplify the notations, we assume for the rest of this chapter that there is only one unobservable event $\tau \in \Sigma \setminus \Sigma_a$. The alphabet of events is then $\Sigma = \Sigma_a \cup \{\tau\}$. The set Σ_a is also partitioned into the set $\Sigma_v \subseteq \Sigma_a$ of the events that may become unobservable at runtime (e.g. the outputs of the system) and the set $\Sigma_{uv} = \Sigma_a \setminus \Sigma_v$ of the events that may not become unobservable to the attacker (e.g. the input actions). To define dynamic projections as in Example 6.1, we first introduce the notion of observability choice which is a mapping from the traces observed by the attacker to the set of events of Σ_a that are observable after this trace. Then, the outcomes of the observability choice always include the set Σ_{uv}.

Definition 6.1 (Observability Choice) *An (observation-based)* observability choice *is a mapping $T : \Sigma_a^* \to \mathcal{P}(\Sigma_a)$ such that for all traces $\mu \in \Sigma_a^*$, $\Sigma_{uv} \subseteq T(\mu)$.*

Definition 6.2 (Dynamic Projection) *An observability choice T uniquely defines an (observation-based)* dynamic projection *by:*

$$\begin{array}{rccl} \pi_T : & \Sigma^* & \to & \Sigma_a^* \\ & \epsilon & \mapsto & \epsilon \\ & w\sigma & \mapsto & \begin{cases} \pi_T(w)\sigma & \text{if } \sigma \in T(\pi_T(w)) \\ \pi_T(w) & \text{otherwise} \end{cases} \end{array}$$

Assuming that the word $w \in \Sigma^*$ is generated by M and $\mu \in \Sigma_a^*$ has been observed by the attacker i.e. $\mu = \pi_T(w)$, then the events that are observable are the ones of $T(\mu)$. In this case, the choice of this set does not change until an observable event occurs in the system. Note that the static projections correspond to observability choices that are constant mappings.

Example 6.2 *The dynamic projection corresponding to the one introduced in Example 6.1 is induced by the observability choice T defined for $\mu \in b^*a$ by $T(\mu) = \{a\}$, and $T(\mu) = \{a, b\}$ for the other observed traces $\mu \in \Sigma^* \setminus b^*a$.*

We denote by *Obs* the set of observability choices, i.e.

$$Obs = \{T : \Sigma_a^* \to \mathcal{P}(\Sigma_a) : \forall \mu \in \Sigma_a^*,\ \Sigma_{uv} \subseteq T(\mu)\}$$

and the set $Obs^\dagger \subseteq Obs$ will represent the ones defining a dynamic projection enforcing the opacity on M. The elements of $Obs^\dagger$ will be called valid observability choices and their associated projections will be called valid projections, i.e. if $T \in Obs^\dagger$ and if $r \in \mathcal{R}(M)$, an attacker cannot infer from the observed trace $\pi_T \circ tr(r)$ whether $lst(r) \in F(\phi)$ for some $\phi \in \Phi$.

Remark 6.1 *We assume in this chapter that the observability depends on the trace observed by the attacker. The techniques proposed below for the synthesis of dynamic projections rely on this assumption. In order to gain in generality, it would be interesting to investigate the problem to other cases where observability depends on the runs, the generated traces or the last states of the runs for example. For such subsequent developments, we will outline in the presentation where this assumption is necessary for the proposed approach.*

For an LTS M as above and a dynamic projection π_T, $\pi_T(\mathcal{L}(M))$ is the set of observed traces.

Definition 6.3 *Given two different observability choices T and T', we say that T and T' are M-equivalent, denoted $T \sim_M T'$, whenever for all $w \in \mathcal{L}(M)$, $\pi_T(w) = \pi_{T'}(w)$.*

The relation $\sim_M$ identifies two observability choices when they define projections that agree on the words of $\mathcal{L}(M)$; they can disagree on other words in Σ^* but since they will not be generated by M, it will not make any difference from the point of view of the attacker. In the sequel we will be interested in computing the interesting part of dynamic projections given M by computing one observability choice in each class of $Obs^\dagger/_{\sim_M}$ since obviously, if $T \sim_M T'$, then T is valid if and only if T' is valid.

For this equivalence relation, we state a lemma that will be useful in the sequel to prove that two observability choices are equivalent.

Lemma 6.1 *Given T, $T' \in Obs$, $T \sim_M T'$ if and only if $\pi_T(\mathcal{L}(M)) = \pi_{T'}(\mathcal{L}(M))$ and for all $\mu \in \pi_T(\mathcal{L}(M))$, $T(\mu) = T'(\mu)$.*

Proof. We just need to remark that as the observability depends on the observed traces, we can reformulate T from π_T and for every $\mu \in \pi_T(\mathcal{L}(M))$ by $T(\mu) = \{\sigma \in \Sigma : \forall w \in \pi_T^{-1}(\mu),\ \pi_T(w\sigma) = \pi_T(w)\sigma\}$. $\square$

In the sequel, we will be interested in checking that M is opaque for a given T and to synthesize dynamic projections enforcing opacity. In Section 6.2, the projection was the natural projection as in the previous chapters and verifying opacity was based on the determinization procedure presented in Chapter 4. Here, we need to find a characterization of these dynamic projections that can be used to check opacity or to enforce it. To do so, we introduce the notion of (dynamic) observer [CT08] that will encode a dynamic projection in terms of transition systems.

Definition 6.4 (Observer) *An* observer *is a tuple $O = (\Sigma_a, X, \delta_o, x_0, V)$ where $(\Sigma_a,\ X,\ \delta_o,\ x_0)$ is a complete and deterministic LTS with X being a (possibly infinite) set of states, $x_0 \in X$ the initial state and $\delta_o : \Sigma_a \times X \to X$ the transition function (a total function). The map $V : X \to \mathcal{P}(\Sigma_a)$ specifies the set of events, with $\Sigma_{uv} \subseteq V(x)$, that the observer keeps observable at state $x \in X$. We require that for all $x \in X$ and for all $\sigma \in \Sigma_a$, if $\sigma \notin V(x)$, then $\delta_o(\sigma, x) = x$, i.e. the observer does not change its state when an unobservable event occurs.*

The last assumption encodes the fact that the observability policy of the dynamic projections depends on the traces observed by the attacker and then cannot change as long as no observable event occurs.

Remark 6.2 *Assuming that the observer is at state x and an event σ occurs, it outputs σ whenever $\sigma \in V(x)$ and then proceeds to $\delta_o(\sigma, x)$. If $\sigma \notin V(x)$, then the observer outputs nothing (ϵ) and remains at state x. An observer can then be interpreted as a special case of functional transducer taking a string $w \in \Sigma^*$ as input, and producing the output which corresponds to the sequence of events it has chosen to keep observable.*

Example 6.3 *Examples of observers are given in Figure 6.3, with $\Sigma = \Sigma_a = \{a, b\}$*

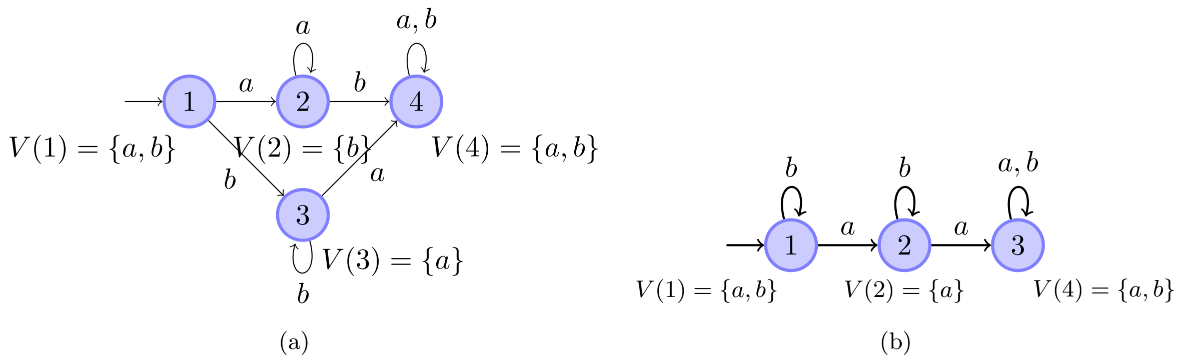


Figure 6.3: Examples of Observers

We now relate the notion of observer to the notion of observability choice.

Proposition 6.1 *If $O = (\Sigma_a, X, \delta_o, x_0, V)$ is an observer, we can define the observability choice T by $T(\mu) = V(\delta_o(\mu, x_0))$.*

Proof. We have then $T : \Sigma_a^* \to \mathcal{P}(\Sigma_a)$ and $\Sigma_{uv} \subseteq T(\mu)$ for all $\mu \in \Sigma_a^*$ as $\Sigma_{uv} \subseteq V(x)$ for all $x \in X$. $\square$

For an observer O, we will denote by $\mathcal{T}(O)$ the observability choice corresponding to the construction given above.

Proposition 6.2 *Given $T \in Obs$, we can construct an observer O such that $\mathcal{T}(O) = T$. This observer is given by $O = (\Sigma_a, \Sigma_a^*, \delta_o, \epsilon, T)$ where for $\mu \in \Sigma_a^*$, $\sigma \in \Sigma_a$, $\delta_o(\sigma, \mu) = \pi_T(\mu\sigma)$. Then O is an observer.*

Proof. The structure $(\Sigma_a, \Sigma_a^*, \delta_o, \epsilon)$ is a complete and deterministic LTS by construction. For a trace $\mu \in \Sigma_a^*$ and $\sigma \in \Sigma_a$, if $\sigma \notin T(\mu)$, then $\pi_T(\mu\sigma) = \pi_T(\mu)$ and then $\delta_o(\sigma, \mu) = \mu$. So O is an observer. $\square$

Note that, like for automata and languages, there might exists several observers encoding the same dynamic projection. For example, the observer depicted in Figure 6.3(b) is one observer that encodes the dynamic projection described in Example 6.2. But, one can consider other observers obtained by unfolding an arbitrary number of times the self-loops in state 1 for example. Note also that the construction above is canonical in the sense that two observability choices defining the same observer are equal. Finally, to mimic the language theory terminology, we will say that $T \in \mathit{Obs}$ is *regular* whenever there exists a finite state observer O such that $\mathcal{T}(O) = T$. Regular observability choices are the ones we will be interested in for practical applications. We will therefore outline how such regular observability choices can be obtained when presenting synthesis techniques for dynamic projections enforcing opacity.

To summarize this part, we can state that with each observability choice T, we can associate an observer O such that $T = \mathcal{T}(O)$. In other words, we can consider an observability choice, a dynamic projection or one of its associated observers whenever one representation is more convenient than the others.

Opacity and Dynamic Projections

A dynamic projection π_T induced by $T \in \mathit{Obs}$ enforces opacity on M if the attacker cannot infer from the observed traces that a run of M reaches a state of $F(\phi)$ for some $\phi \in \Phi$. Then, we should be able to compute the state estimates as for the construction of chapter 4 and a secret ϕ will be disclosed when such a state estimate is completely included in $F(\phi)$. But, for technical reasons that will appear in Section 6.3, we consider instead the condensed state estimates as in Chapter 5. Then, we define Estim_T mapping an observed trace $\mu \in \Sigma_a^*$ to the set of states that the system reaches directly after the occurrence of the last observed event of μ:

$$\begin{array}{rrcl} \mathit{Estim}_T : & \Sigma_a^* & \to & \mathcal{P}(Q) \\ & \epsilon & \mapsto & Q_0 \\ & \mu\sigma & \mapsto & \{q \in Q : \exists r \in \mathcal{R}(M),\ r = q_0 \overset{w\sigma}{\to} q \wedge \pi_T(w\sigma) = \mu\sigma\} \end{array}$$

The following proposition gives an alternative formulation of Estim_T, showing that it can also be defined by induction.

Proposition 6.3 *Given $T \in \mathit{Obs}$, for all $\mu \in \Sigma_a^*$ and all $\sigma \in \Sigma_a$, $\mathit{Estim}_T(\mu\sigma) = \mathit{post}_M(\{\sigma\}) \circ \mathit{reach}_M(\Sigma \setminus T(\mu))(\mathit{Estim}_T(\mu))$.*

Proof. The proof, by induction on the length of μ, is similar to the proof on Proposition 4.1. □

Based on the definition of *Estim*, we can express with the following proposition the set of observed traces disclosing a secret.

Proposition 6.4 *Let $T \in Obs$ and $\phi \in \Phi$. Then,*

$$DTraces(\mathcal{R}(M), \pi_T \circ tr)(\phi) = \{\mu \in \Sigma_a^* : reach_M(\Sigma \setminus T(\mu))(Estim_T(\mu)) \subseteq F(\phi)\}$$

Next, we will investigate how to verify whether $DTraces(\mathcal{R}(M), \pi_T \circ tr)(\phi) = \emptyset$, i.e. that the dynamic projection π_T enforces opacity.

Checking Opacity for Regular Projections

The problem we are going to address consists in checking whether a given regular dynamic projection enforces opacity and we show that this problem is PSPACE-complete with respect to the size of M and the observer. This problem is stated as follows:

Problem 6.2 (Dynamic Opacity Problem)

- ***Input:*** *A finite LTS $M = (\Sigma, Q, \delta, Q_0)$, a finite set of state-based secret predicates Φ and a regular dynamic projection π_T.*
- ***Problem:*** *Is M opaque for π_T ?*

Let $O = (\Sigma_a, X, \delta_o, x_0, V)$ be an observer such that $\mathcal{T}(O) = T$. At the beginning, we do not need to assume that O is finite. We construct an LTS which represents what an attacker will see under the dynamic choices of observable events made by T. This construction simply replaces the events by the unobservable event τ when an event is not observable according to T. To do so, we define the LTS

$$M \otimes O = (\Sigma, Q \times X, (q_0, x_0), \delta_{M \otimes O})$$

where $\delta_{M\otimes O}$ is defined for each $(q, x) \in Q \times X$ by:

- if $\sigma \in V(x)$ then $\delta_{M\otimes O}(\sigma, (q, x)) = \delta(\sigma, q) \times \{\delta_o(\sigma, x)\}$;
- $\delta_{M\otimes O}(\tau, (q, x)) = \big(\cup_{\sigma \in \Sigma \setminus V(x)} \delta(\sigma, q)\big) \times \{x\}$.

Then, we define the set of state-based predicates Φ_o over $E(\Sigma, Q \times X)$ by $F(\phi_o) = F(\phi) \times X$ for each secret predicate $\phi \in \Phi$. Recall also that $\pi_a = \pi_{\Sigma \to \Sigma_a}$. We can now state that this transformation preserves the set of observations disclosing a secret.

Proposition 6.5 *For all $\phi \in \Phi$,*

$$DTraces(\mathcal{R}(M), \pi_T \circ tr)(\phi) = DTraces(\mathcal{R}(M \otimes O), \pi_a \circ tr)(\phi_o)$$

Proof. As O is complete and deterministic, we claim that for all $r = q_0 \xrightarrow{w} q \in \mathcal{R}(M)$ with $\pi_T(w) = \mu$, there exists a unique $r_o \in \mathcal{R}(M \otimes O)$, $r_o = (q_0, x_0) \xrightarrow{u} (q, x)$ such that $\pi_a(u) = \mu$. To prove this, we reason by induction on the length of w. Suppose that this holds for all $w \in \Sigma^n$. Let $r = q \xrightarrow{w} q' \xrightarrow{\sigma} q \in \mathcal{R}(M)$ with $\mu = \pi_T(w)$. Then, there exists a unique $r_o = (q_0, x_0) \xrightarrow{u} (q', x') \in \mathcal{R}(M \otimes O)$ such that $\pi_a(u) = \mu$. If $\sigma \notin V(x')$, then $\pi_T(w\sigma) = \pi_T(w) = \mu$ and $r_o = (q_0, x_0) \xrightarrow{u} (q', x') \xrightarrow{\tau} (q, x')$ is the candidate as $\delta_o(\sigma, x') = x'$ and $\pi_a(u\tau) = \pi_a(u) = \mu$. Now, if $\sigma \in V(x')$, then $\pi_T(w\sigma) = \mu\sigma$ and $r_o = (q_0, x_0) \xrightarrow{u} (q', x') \xrightarrow{\sigma} (q, \delta_o(\sigma, x'))$ is the candidate as $V(x') \subseteq \Sigma_a$ implies that $\sigma \in \Sigma_a$ and then $\pi_a(u\sigma) = \mu\sigma$. Then, the proposition follows from the fact that $q \in F(\phi)$ if and only if $(q, x) \in F(\phi_o)$. $\square$

Applying this result, we can relate the opacity of M for π_T with the Φ_o-opacity of $M \otimes O$ for π_a.

Corollary 6.1 *The LTS M is Φ-opaque for π_T if and only if $M \otimes O$ is Φ_o-opaque for π_a.*

Note that we did not need to assume that O is a finite state observer to establish the Proposition 6.5 and Corollary 6.1. By consequence this result holds for any kind of observer. For verification purpose, assume now that O is a finite state observer. Then, applying Corollary 6.1, we can state the following Theorem to solve Problem 6.2.

Theorem 6.2 *For a regular dynamic projection, Problem 6.2 is PSPACE-complete.*

Proof. Verifying that $M \otimes O$ is Φ_o-opaque is PSPACE-complete in the size of $M \otimes O$ according to Theorem 4.2. Since computing $M \otimes O$ is polynomial in the size of M and O, the Theorem follows corollary 6.1. $\square$

6.3 Enforcing Opacity with Dynamic Projections

We have seen in the previous section how to verify opacity for dynamic projections when an observer encoding this projection was provided. In this section, we will be interested in *synthesizing* dynamic projections enforcing opacity.

For the controller synthesis problem of chapter 5, the set of solutions, i.e. the set of controlled languages enforcing opacity was closed under arbitrary union. This implied the existence of a unique supremal solution and we presented algorithms to compute this supremal solution. But, unfortunately, the following remark states that union does not preserve the validity of the observability choices.

Remark 6.3 *The set of valid dynamic projections $Obs^{\dagger}$ is not closed under union. Indeed, consider the LTS M of Figure 6.4 where $\Sigma = \Sigma_a = \{a, b\}$ and two state-based secret predicates ϕ_1 and ϕ_2 defined by $F(\phi_1) = \{q_1\}$ and $F(\phi_2) = \{q_2\}$. We define the two observability choices T_1 and T_2 by*

$$T_1(\epsilon) = \{a\} \text{ and } T_1(\mu) = \Sigma_a \text{ if } |\mu| \geq 1$$

$$T_2(\epsilon) = \{b\} \text{ and } T_2(\mu) = \Sigma_a \text{ if } |\mu| \geq 1$$

Then, both T_1 and T_2 define valid dynamic projections. Define now $T = T_1 \cup T_2$. Then $T(\mu) = \Sigma_a$ for all $\mu \in \Sigma_a^*$ and such an observability choice does not enforce the opacity on M, i.e. $T \notin Obs^{\dagger}$. For example, the run $q_0 \xrightarrow{a} q_1$ is the only run explaining the observed trace a and the secret ϕ_1 is then disclosed.

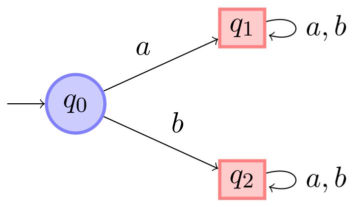


Figure 6.4: The set $Obs^{\dagger}$ is not closed by union

By consequence, we cannot obtain a supremal solution with respect to the order relation defined by the inclusion over the maps of Obs. In this thesis, we do not present possible order relations with realistic applications that will also imply the existence of a unique supremal solution. As for the work on controller synthesis, a possible extension of this work will be to enforce a security policy consisting for example in confidentiality and availability requirements. Towards such subsequent developments, we propose in this chapter a method to represent with a finite game the set of valid dynamic projections. The set of positional strategies on this games will correspond to the set of regular observability choices enforcing opacity. The problem we are interested in is stated as follows:

Problem 6.3 (Dynamic Projection Synthesis Problem)

- ***Input***: *A finite LTS $M = (\Sigma, Q, \delta, Q_0)$ and a finite set Φ of state-based secret predicates.*
- ***Problems:***
  - ***(A)***: *Decide whether the set of regular observability choices $Obs^{\dagger}$ is empty ?*
  - ***(B)***: *Compute the set of regular observability choices $Obs^{\dagger}$.*

Remark 6.4 *Our aim is actually to be able to generate at least one observer for each representative of* $Obs^\dagger/_{\sim_M}$*, thus capturing all the interesting dynamic projections.*

We start by giving the complexity of solving Problem 6.3 (A).

Proposition 6.6 *The problem 6.3 (A) is PSPACE-complete.*

Proof. Let the dynamic projection T_0 defined by $T_0(\mu) = \Sigma_{uv}$ for all $\mu \in \Sigma^*$. The projection π_{T_0}, always hiding the events of Σ_v, is then the most imprecise projection that can be defined. Indeed, for $T \in Obs$ and for all $w, w' \in \Sigma_a^*$, $\pi_T(w) = \pi_T(w')$ implies $\pi_{T_0}(w) = \pi_{T_0}(w')$. Then, according to Proposition 3.3, M is opaque for π_T implies that M is opaque for π_{T_0}. So checking whether $T_0 \in Obs^\dagger$ provides a necessary and sufficient condition to the existence of a valid observability choice. Since checking whether M is opaque with respect to T_0 is PSPACE-complete according to Theorem 6.2, this is also the complexity of Problem 6.3 (A). □

Note that T_0 is regular as it can be encoded by a one state observer. So the opacity of M for π_{T_0} also implies that the set of regular observability choices of $Obs^\dagger$ is not empty.

6.3.1 Reduction to a 2-player Safety Game

To solve Problem 6.3 (B), we reduce it to a safety 2-player game. Player 1 will play the rôle of an observability choice and Player 2 the rôle of the system deciding what the attacker observes. Assume that according to the attacker and the trace that have been observed, M may have reached $e = \{q_1, q_2, \dots, q_n\}$ directly after the last observed event[2]. A round in the game is: given such an estimate e, Player 1 chooses which letters should be observable next i.e. a set $t \subseteq \Sigma_a$ such that $\Sigma_{uv} \subseteq t$; then, it hands it over to Player 2 who picks up an observable letter $\sigma \in t$; this determines a new set of states that M may have reached directly after σ, and the turn is back to Player 1. The goals of the Players are defined by:

- The goal of Player 2 is to pick up a sequence of letters such that the set of states that can be reached after this sequence and all subsequent unobservable trajectories is included in one of the $F(\phi)$ for $\phi \in \Phi$. If Player 2 can do this, then the secret ϕ is disclosed to the attacker. Player 2 thus plays a *reachability game* trying to enforce a particular set of states, say *Bad* (i.e. the states in which the secret is disclosed).

- The goal of Player 1 is opposite. It must keep the game in a safe set of states where the secret is not disclosed. Thus Player 1 plays a *safety game* trying to keep the game in the complement set of *Bad*.

[2]We consider here the condensed state estimate, like in the construction of 5.3.2.

Since we are playing a (finite) turn-based game, Player 2 has a strategy to enforce *Bad* iff Player 1 has no strategy to keep the game in the complement set of *Bad* (turn-based finite games are *determined*, consequence of [Mar75]).

We now formally define the 2-player game and show how to obtain a finite representation of all the regular valid dynamic projections from this game. Let $H = (\Upsilon_1 \cup \Upsilon_2, S_1 \cup S_2, \delta_H, Q_0)$ be a deterministic game LTS given by:

- Player 1 chooses a set of events to hide in $\Sigma_v \subseteq \Sigma_a$. Thus, the events of Player 1 are in the alphabet $\Upsilon_1 = \{t \subseteq \Sigma_a : \Sigma_{uv} \subseteq t\}$ and Player 2 choses the next observed event, therefore $\Upsilon_2 = \Sigma_a$;
- $S_1 = \mathcal{P}(Q)$ is the set of Player 1 states and $S_2 = \mathcal{P}(Q) \times \Upsilon_1$ the set of Player 2 states;
- the initial state of the game is the Player 1 state Q_0, i.e. the set of initial states of M;
- the transition relation $\delta_H \subseteq (S_1 \times \Upsilon_1 \times S_2) \cup (S_2 \times \Upsilon_2 \times S_1)$ is given by:
  - Player 1 moves (choice of the observable events): if $e \in S_1$, $t \in \Upsilon_1$, then $\delta_H(t, e) = (e, t)$;
  - Player 2 moves (choice of the next observed event): if $(e, t) \in S_2$, $\sigma \in t$ and $e' = \mathit{post}_M(\{\sigma\}) \circ \mathit{reach}_M(\Sigma \setminus t)(e) \neq \emptyset$, then $\delta_H(\sigma, (e, t)) = e'$.

The set of states *Bad* is defined by:

$$\mathit{Bad} = \{(e, t) \in S_2 : \exists \phi \in \Phi,\ \mathit{reach}(\Sigma \setminus t)(e) \subseteq F(\phi)\}$$

Remark 6.5 *The fact that the observability choice depends on the trace observed by the attacker is important for this definition of Bad. Indeed, when the attacker observes an trace μ which brings the game is at state (e, t) (we will see later how is the connection between μ and (e, t)), then the attacker knows which events are observable after μ, i.e. the events of t. And this set will not change until an event of t, i.e. observable, occurs in M. Therefore, the set $\mathit{reach}(\Sigma \setminus t)(e)$ is exactly the set of states that M may have reached after μ. If the observability choice depends on the word generated by M, then the computation of this state estimate is more complicated as the attacker is not aware of all the changes in the observability of events.*

Let $\mathcal{R}_i(H)$, $i = 1, 2$ be the set of runs of H ending in a Player i state. A *strategy* for Player i is a mapping $f_i : \mathcal{R}_i(H) \to \Upsilon_i$ that associates with each run ending in a Player i

state, the new choice of Player i. Given two strategies f_1 and f_2, the game H generates the set of runs $\mathit{Outcome}(f_1, f_2, H)$ combining the choices of Player 1 and Player 2 given by f_1 and f_2. Let $\mathit{Outcome}_1(f_1, H) = (\cup_{f_2}\mathit{Outcome}(f_1, f_2, H)) \cap \mathcal{R}_1(H)$ denote the set of runs ending in a Player 1 state which can be generated in the game when Player 1 plays f_1 against all the possible strategies of Player 2. The set of runs $\mathit{Outcome}_2(f_2, H)$ is similarly defined. We can then identify two strategies when they differ only on runs that are not reachable by playing these strategies. Then, for $i = 1, 2$, we say that two strategies f, f' of Player i are equivalent with respect to the game automaton H, denoted $f \sim_H f'$ if $\mathit{Outcome}_i(f, H) = \mathit{Outcome}_i(f', H)$. Note that this implies that for all runs $\rho \in \mathit{Outcome}_i(f, H)$, $f(\rho) = f'(\rho)$. Finally, we denote by Strat_i the set of strategies (modulo $\sim_H$) for Player i, i.e. $\mathit{Strat}_i = \{f_i : \mathcal{R}_i(H) \mapsto \Upsilon_i\}/_{\sim_H}$. For simplification, we will identify, in the sequel the map of $\mathcal{R}_i \to \Upsilon_i$ and the elements of Strat_i. We will just need to prove the equality of two strategies $f, f' : \mathcal{R}_i \to \Upsilon_i$ by $\mathit{Outcome}_i(f, H) = \mathit{Outcome}_i(f', H)$.

The strategy $f_1 \in \mathit{Strat}_1$ is a *winning strategy* for Playing 1 in H for avoiding Bad if for all $f_2 \in \mathit{Strat}_2$, no run of $\mathit{Outcome}(f_1, f_2, H)$ contains a state of Bad. A winning strategy for Player 2 is a strategy $f_2 \in \mathit{Strat}_2$ such that for every strategy $f_1 \in \mathit{Strat}_1$ of Player 1, every run $\rho \in \mathit{Outcome}(f_1, f_2, H)$ can be be extended to a run of $\mathit{Outcome}(f_1, f_2, H)$ reaching a state of Bad.

We have seen that with the game defined above, either Player 1 has a winning strategy or Player 2 has a winning strategy. The purpose of defining this game is to show that the set of valid observability choices $\mathit{Obs}^\dagger$ corresponds to the set of winning strategies of Player 1, thus solving Problem 6.3 (B) by obtaining a finite game LTS representing the set $\mathit{Obs}^\dagger$.

Informal Presentation of the Reduction

We now give the general ideas of the construction presented below. The first step is to establish a bijective correspondence between the set Obs of observability choices and the set Strat_1 of strategies of Player 1. Let us first define $p_i = \pi_{\Upsilon \to \Upsilon_i} \circ tr$, $i = 1, 2$, where $\Upsilon = \Upsilon_1 \cup \Upsilon_2$. The maps p_i project every run of $\mathcal{R}(H)$, to the sequence of events of Υ_i occurring in the run. To establish such a correspondence, we associate every observability choice to a strategy of Player 1 via the map α defined as follows.

Definition 6.5 *The map α associates to each observability choice $T \in \mathit{Obs}$ a strategy of Player 1 which consists in playing the outcome of T according to the trace observed by the attacker, i.e. the outcomes of the projection π_2. Formally,*

$$\begin{array}{rcccl} \alpha : & \mathit{Obs} & \to & \mathcal{R}_1(H) & \to \Upsilon_1 \\ & T & \mapsto & \rho & \mapsto T(p_2(\rho)) \end{array}$$

But, the definition of the game LTS H depends on the system M whereas the observability choices are given as maps from the whole set Σ_a^* and thus, a priori, independently of H. We have to show that the game defined above faithfully represents the effects of dynamic projections with respect to an inquisitive attacker trying to infer the truth of some predicate of Φ on the runs of M. We show then that α establishes a correspondence between the set of strategies of Player 1 and the set of observability choices modulo $\sim_M$. For this, we show with Proposition 6.7 that two observability choices $T, T' \in \mathit{Obs}$ define the same strategy if and only if the induced dynamic projections agree on the words generated by M, i.e.

$$\mathit{Outcome}_1(\alpha(T), H) = \mathit{Outcome}_1(\alpha(T'), H) \iff T \sim_M T'$$

In that case, the relation $\sim_M$ corresponds exactly to the equivalence kernel from Definition 2.4. So, applying the proposition 2.1, the canonical quotient map $\mathit{can}(\alpha)$ is such that the following diagram is commutative:

$$\begin{tikzcd}
\mathit{Obs} \arrow[r, "\alpha"] \arrow[d, "{[\cdot]_{\sim_M}}"'] & \mathit{Strat}_1 \\
\mathit{Obs}/_{\sim_M} \arrow[ur, "\mathit{can}(\alpha)"'] &
\end{tikzcd}$$

We also know from Proposition 2.1 that the map $\mathit{can}(\alpha) : \mathit{Obs}/_{\sim_M} \to \alpha(\mathit{Obs})$ is injective. Then we need to show that α is surjective. For this, we define the map β associating each strategy of Player 1 to a (partial) function $\Sigma_a^* \to \mathcal{P}(\Sigma_a)$. For this, given a strategy f, we show that for each $\mu \in \Sigma_a^*$, there exists at most one run $\rho \in \mathit{Outcome}_1(f, H)$ such that $\pi_2(\rho) = \mu$. When such a run exists, we denote it by $\rho_{f,\mu}$. Using this, we can define the map β as follows.

Definition 6.6 *The map β associates to each strategy $f \in \mathit{Start}_1$ an observability choice defined by the moves of f for the observed traces of $\pi_2(\mathit{Outcome}_1(f, H))$ and by Σ_a otherwise.*

$$\begin{array}{llllll}
\beta : & \mathit{Strat}_1 & \to & \Sigma_a^* & \to & \mathcal{P}(\Sigma_a) \\
& f & \mapsto & \mu & \mapsto & \begin{cases} f(\rho_{f,\mu}) \text{ when } \rho_{f,\mu} \text{ exists} \\ \Sigma_a \text{ otherwise} \end{cases}
\end{array}$$

We show with Lemma 6.3 that with this definition of β, $\alpha \circ \beta = id_{\mathit{Strat}_1}$, i.e. that we retrieve exactly the strategy f by applying α to the partial function $\beta(f)$. Then, this implies that α is surjective, and by consequence that $\mathit{can}(\alpha)$ establishes a bijective correspondence between $\mathit{Obs}/_{\sim_M}$ and Strat_1.

The second step is to show, with Proposition 6.10, that an observability choice defines, by applying α, a winning strategy on H if and only if this observability choice is valid. This establishes then a bijective correspondence between the set of valid observability choices (modulo $\sim_M$) and the set of winning strategies on H.

Finally, the last step is to show that the set of winning strategies of Player 1 can be represented by restricting the arena H to safe moves, i.e. to moves that prevent Player 2 to win the game.

Technical Developments

We will now more formally present the construction discussed above. We start with a technical lemma that will be useful to define the map β.

Lemma 6.2 *Given a strategy $f \in \mathit{Strat}_1$, for all $\mu \in \Sigma^*$, there exists at most one run $\rho \in \mathit{Outcome}_1(f, H)$ such that $p_2(\rho) = \mu$.*

Proof. For a run $\rho \in \mathit{Outcome}_1(f, H)$ with $\mathit{lst}(\rho) = e$, the strategy f uniquely defines the next state (e, t) where $t = f(\rho)$. Also, The LTS H being deterministic, given $\sigma \in \Sigma_a$, there is at most one run ρ' such that $p_2(\rho') = p_2(\rho)\sigma$ and this run is $\rho \xrightarrow{t} (e, t) \xrightarrow{\sigma} \delta_H(\sigma, (e, t))$ which exists if $\delta_H(\sigma, (e, t))$ is defined. The proof follows by induction. □

This lemma proves that the construction of β from Definition 6.6 is effectively possible. We directly apply this lemma to prove that $\alpha \circ \beta = id_{\mathit{Strat}_1}$, which implies that α, from Definition 6.5 is a surjective map, i.e. that every strategy of Player 1 is the image by α of an observability choice.

Lemma 6.3 *For all $f \in \mathit{Strat}_1$, $\alpha \circ \beta(f) = f$.*

Proof. Let $f \in \mathit{Strat}_1$. Then,

$$\begin{aligned}
\alpha \circ \beta(f)(\rho) &= \alpha(\beta(f))(\rho) \\
&= \beta(f)(p_2(\rho)) \text{ by definition of } \alpha \\
&= f(\rho) \text{ by definition of } \beta \text{ as } \rho \text{ is the only run} \\
&\qquad \text{of } \mathit{Outcome}_1(f, H) \text{ with trace } p_2(\rho) \text{ (Lemma 6.2)}
\end{aligned}$$

The lemma is then established. □

We show now that two observability choices define the same strategy of Player 1 if and only if the induced dynamic projections agree on the words of $\mathcal{L}(M)$. The first lemma states that the game preserves the (condensed) state estimates.

Lemma 6.4 *Given $T \in Obs$, for all $\rho \in Outcome_1(\alpha(T), H)$,*

$$lst(\rho) = Estim_T(p_2(\rho))$$

Proof. We prove this by induction on $|p_2(\rho)|$. If $|p_2(\rho)| = 0$, then $\rho = Q_0 = Estim_T(\epsilon)$. Suppose now that the lemma holds if $|p_2(\rho)| = n$ and let $\rho' = \rho \xrightarrow{t} (e, q) \xrightarrow{\sigma} e' \in Outcome_1(\alpha(T), H)$. We have then $e = Estim_T(\mu)$ where $\mu = p_2(\rho)$. According to the definition of α, $t = T(\mu)$. According to the definition of δ_H,

$$\begin{aligned} e' &= post_M(\{\sigma\}) \circ reach_M(\Sigma \setminus t)(e) \\ &= post_M(\{\sigma\}) \circ reach_M(\Sigma \setminus T(\mu))(Estim_T(\mu)) \\ &= Estim_T(\mu\sigma) \text{ according to Proposition 6.3} \end{aligned}$$

which establishes the lemma. □

This result implies that the observed traces generated by the game, the outcomes of π_2, are also traces observed by the attacker. We show next that the game exactly generates the observed traces of $\pi_T(\mathcal{L}(M))$.

Lemma 6.5 *Given $T \in Obs$, $p_2(Outcome_1(\alpha(T), H)) = \pi_T(\mathcal{L}(M))$.*

Proof. As $Estim_T(\mu) \neq \emptyset$ implies $\mu \in \pi_T(\mathcal{L}(M))$, it follows from Lemma 6.4 that $p_2(Outcome_1(\alpha(T), H)) \subseteq \pi_T(\mathcal{L}(M))$. We prove the other inclusion by induction on the length of $\mu \in \pi_T(\mathcal{L}(M))$. If $\mu = \epsilon$, $Estim_T(\epsilon) = Q_0 \in Outcome_1(\alpha(T), H)$. Suppose that for all $\mu \in \Sigma_a^n \cap \pi_T(\mathcal{L}(M))$, $\mu \in p_2(Outcome_1(\alpha(T), H))$, and let $\sigma \in \Sigma_a$ such that $\mu\sigma \in \pi_T(\mathcal{L}(M))$. By hypothesis, there exists a run $\rho \in Outcome_1(\alpha(T), H)$ such that $p_2(\rho) = \mu$. Then, let $e = lst(\rho)$. As $\mu\sigma \in \pi_T(\mathcal{L}(M))$, $\sigma \in T(\mu)$ and $Estim_T(\mu\sigma) \neq \emptyset$. According to Lemma 6.4 $e = Estim_T(\mu)$. Then, $e' = post_M(\{\sigma\}) \circ reach_M(\Sigma \setminus T(\mu))(e) = Estim_T(\mu\sigma) \neq \emptyset$. So $\delta_H(\sigma, (e, T(\mu)))$ is defined and $\rho' = \rho \xrightarrow{T(\mu)} (e, T(\mu)) \xrightarrow{\sigma} e' \in Outcome_1(\alpha(T), H)$. As $p_2(\rho') = \mu\sigma$, the lemma is established. □

We will now use these two lemmas above to prove that computing the quotient of α by $\sim_M$ provides the canonical quotient map of α. In other words, two observability choices inducing dynamic projections which agree on the words generated by M, will define, via α, the same strategy of Player 1.

Proposition 6.7 *Given $T, T' \in Obs$,*

$$T \sim_M T' \iff Outcome_1(\alpha(T), H) = Outcome_1(\alpha(T'), H)$$

Proof. Suppose that $T \sim_M T'$. We show that $\mathit{Outcome}_1(\alpha(T), H) = \mathit{Outcome}_1(\alpha(T'), H)$ by induction on the length of the runs. Suppose that this holds if $|p_2(\rho)| = n$ and let $\rho' = \rho \xrightarrow{t} (e,t) \xrightarrow{\sigma} e' \in \mathit{Outcome}_1(\alpha(T), H)$. Then, $t \in \alpha(T)(\rho) = T(p_2(\rho))$. According to Lemma 6.5, $p_2(\rho) \in \pi_T(\mathcal{L}(M))$. Then, following Lemma 6.1, $T(p_2(\rho)) = T'(p_2(\rho)) = \alpha(T')(\rho)$. So $t \in \alpha(T')(\rho)$ and $\rho' \in \mathit{Outcome}_1(\alpha(T'), H)$. The base case $\rho = \epsilon$ is trivial. As T and T' play a symmetrical rôle, we obtain $\mathit{Outcome}_1(\alpha(T), H) = \mathit{Outcome}_1(\alpha(T'), H)$.

Suppose now that T and T' are such that $\mathit{Outcome}_1(\alpha(T), H) = \mathit{Outcome}_1(\alpha(T'), H)$. Then $\pi_T(\mathcal{L}(M)) = p_2(\mathit{Outcome}_1(\alpha(T), H)) = p_2(\mathit{Outcome}_1(\alpha(T'), H)) = \pi_{T'}(\mathcal{L}(M))$. According to Lemma 6.1, it remains to show that for all $\mu \in \pi_T(\mathcal{L}(M))$, $T(\mu) \neq T'(\mu)$. To raise a contradiction, suppose that for some $\mu \in \pi_T(\mathcal{L}(M))$, $T(\mu) \neq T'(\mu)$. Then, we can find $\rho \in \mathit{Outcome}_1(\alpha(T), H)$ such that $p_2(\rho) = \mu$. As $\mu \in p_2(\mathit{Outcome}_1(\alpha(T'), H))$, applying Lemma 6.2, ρ is also the unique run in $\mathit{Outcome}_1(\alpha(T'), H)$ such that $p_2(\rho) = \mu$. Then $\alpha(T)(\rho) = T(\mu) \neq T'(\mu) = \alpha(T')(\rho)$. Let $e = \mathit{lst}(\rho)$ and $t \in \alpha(T)(\rho) \setminus \alpha(T')(\rho)$ (or $t \in \alpha(T')(\rho) \setminus \alpha(T)(\rho)$, the rest of the proof is symmetrical w.r.t T and T'). Then, if $\sigma \in t$ such that $e' = \delta(\sigma, (e,t))$, $\rho \xrightarrow{t} (e,t) \xrightarrow{\sigma} e' \in \mathit{Outcome}_1(\alpha(T), H) \setminus \mathit{Outcome}_1(\alpha(T'), H)$. So $\mathit{Outcome}_1(\alpha(T), H) \neq \mathit{Outcome}_1(\alpha(T'), H)$ which is a contradiction. Then, we must have $T(\mu) = T'(\mu)$ for all $\mu \in \pi_T(\mathcal{L}(M))$. Finally, $T \sim_M T'$. □

Proposition 6.8 *The quotient map $\mathit{can}(\alpha)$ is a bijective correspondence between the set of observability choices (modulo $\sim_M$) and the set of strategies of Player 1.*

Proof. Following Propositions 2.1 and 6.7, the map

$$\begin{array}{rccl} \mathit{can}(\alpha): & \mathit{Obs}/_{\sim_M} & \to & \mathit{Strat}_1 \\ & [T]_{\sim_M} & \mapsto & \alpha(T) \end{array}$$

is such that $\alpha = \mathit{can}(\alpha) \circ [\cdot]_{\sim_M}$. Furthermore, this map is also injective. As $\alpha \circ \beta = \mathit{id}_{\mathit{Strat}_1}$, the map α is surjective. So $\mathit{can}(\alpha)$ is also surjective and establishes then a bijection between $\mathit{Obs}/_{\sim_M}$ and Strat_1. □

We can now establish a link between the winning strategies of Player 1 and the dynamic projections enforcing opacity.

Proposition 6.9 *Given $T \in \mathit{Obs}$, $T \in \mathit{Obs}^\dagger$ if and only if $\alpha(T)$ is a winning strategy for Player 1 in H.*

Proof. Assume that $T \in \mathit{Obs}^\dagger$ and let $\rho \in \mathit{Outcome}_1(\alpha(T), H)$ with $e = \mathit{lst}(\rho)$ and $\mu = p_2(\rho)$. Then, according to Lemma 6.4, $e = \mathit{Estim}_T(\mu)$. Since T is a valid observability choice, for all $\phi \in \Phi$, $reach_M(\Sigma \setminus T(\mu))(e) \nsubseteq F(\phi)$; so $(e, T(\mu)) \notin Bad$. This implies that $\alpha(T)$ is a winning strategy.

For the other implication, assume that $T \notin Obs^{\dagger}$. This means that there exists $\phi \in \Phi$ and a trace $\mu \in \pi_T(\mathcal{L}(M))$ such that $reach_M(\Sigma \setminus T(\mu))(Estim_T(\mu)) \subseteq F(\phi)$. Since $\mu \in \pi_T(\mathcal{L}(M))$, then according to Lemmas 6.5 and 6.2, there exists a unique run $\rho \in Outcome_1(\alpha(T), H)$ such that $p_2(\rho) = \mu$. Let $e = lst(\rho) = Estim_T(\mu) \in Bad$. Then $T(\mu)$ is the only move that Player 1 can play after ρ following the strategy $\alpha(T)$ and $(e, T(\mu)) \in Bad$ so $\alpha(T)$ is not a winning strategy. □

Theorem 6.3 *The map $can(\alpha)$ establishes a bijective correspondence between the set of valid observability choices (modulo $\sim_M$) and the set of winning strategies of Player 1 in H.*

So, based on this Theorem, we will see how to represent the set of winning strategies of Player 1 which will provide a finite representation of the set of valid dynamic projections.

6.3.2 The Set of Valid Dynamic Projections

We will now see how to represent the set of valid dynamic projections and also how to exhibit regular projections.

Theorem 6.4 *The set of valid observability choices (modulo $\sim_M$) can be represented by a finite automaton.*

Proof. As H is a turn-based 2-player game under full observation, we can compute the set of winning strategies that are based on H (see [Tho95]). It is defined as follows: Let us first compute the set of winning states of the game for player 1. For this, let $Good = (S_1 \cup S_2) \setminus Bad$ be the set of safe states of H. To solve this 2-player game, we define the $Cpre$ operator by:

$$\begin{aligned} Cpre(X) = \quad & \{e \in S_1 : \exists t \in \Upsilon_1, \delta_H(t, e) \in X\} \\ & \cup \{(e, t) \in S_2 : \forall \sigma \in t, \delta_H(\sigma, (e, t)) \in X\} \end{aligned}$$

The operator $Cpre$ is monotone on the lattice $\mathcal{P}(S_1 \cup S_2)$. Then, by computing the greatest fixpoint $gfp(X \mapsto Good \cap Cpre(X))$, we obtain the set $Win = \cap_i Cpre^i(Good)$ of winning states of the game for Player 1 [Tho95]. As the set of states is finite, this computation terminates. If the initial state of the game belongs to $Win = \cap_i Cpre^i(Good)$, then there is a strategy for Player 1 to win.

Consider now the following finite LTS $\mathcal{H}$ derived from H and defined by $\mathcal{H} = (\Upsilon_1 \cup \Upsilon_2$, Win, $\delta_{\mathcal{H}}$, $Q_0)$, where $\delta_{\mathcal{H}}$ is the restriction of δ_H to the states Win, i.e. $\delta_{\mathcal{H}}$ is undefined whenever δ_H is undefined or the image is a state outside Win. Now, f is a winning strategy for Player 1 w.r.t. H and Bad if and only if for any run $\rho \in \mathcal{R}_1(H)$, the move $f(\rho)$ follows the restriction $\delta_{\mathcal{H}}$, namely, every move defined by f is a move of $\mathcal{H}$. Now, from Theorem 6.3, given a winning strategy f, we can define a valid observability choice which is encoded by f and H. $\square$

Remark 6.6 *Note that, according to Theorem 6.3 and Proposition 6.6, the opacity of M for the static projection $\pi_{\Sigma \to \Sigma_{uv}}$ is also a necessary and sufficient condition for the existence of a winning strategy for Player 1.*

The previous theorem states that $\mathcal{H}$ can be used to generate any observer. We will see with the next proposition how to define a regular observability choice considering a state-based winning strategy.

Proposition 6.10 *Given a winning strategy $f \in Strat_1$ that is state based, we can define a finite state observer encoding the corresponding valid dynamic projection.*

Proof. If f is state-based, then there exists a map $\bar{f} : S_1 \to \Upsilon_1$ such that for all run $\rho \in \mathcal{R}_1(H)$, $f(\rho) = \bar{f} \circ lst(\rho)$. Then, the corresponding observability choice can be implemented by the observer $O = (\Sigma_a, S_1 \cup \{x_{new}\}, \delta_o, Q_0, V)$ where for $e \in S_1$, $\delta_o(\sigma, e) = e'$ if $e' = \delta_H(\sigma, (e, \bar{f}(e)))$ is defined and $\delta_o(\sigma, e) = x_{new}$ otherwise. We complete the definition of δ_o by $\delta_o(\sigma, x_{new}) = x_{new}$. Also, V is defined by $V(e) = \bar{f}(e)$ and $V(x_{new}) = \Sigma_a$. $\square$

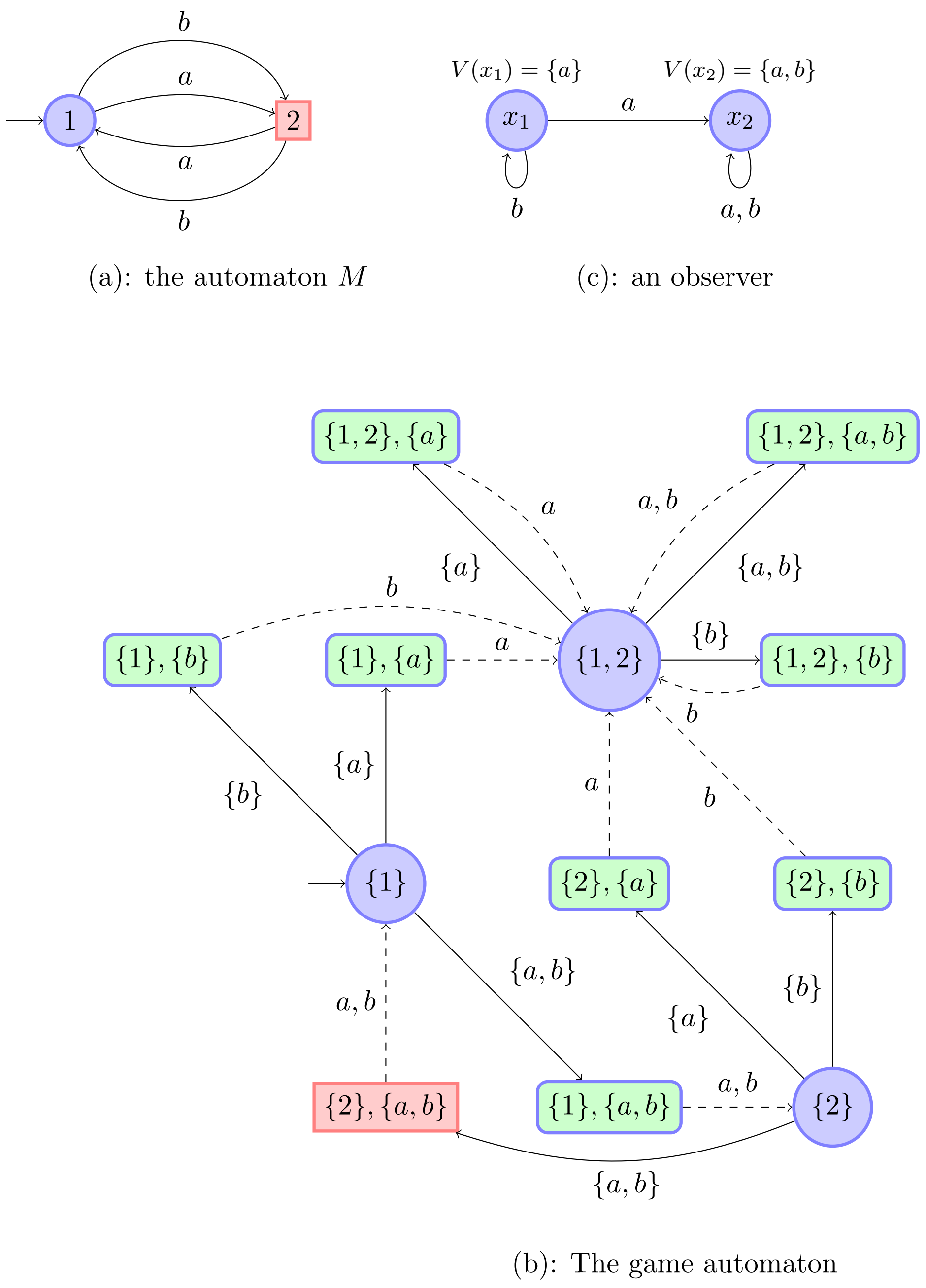
Finally, we can now state an upper bound for the complexity of Problem 6.3. Indeed, an immediate corollary of Theorem 6.4 is the following:

Corollary 6.2 *Problem 6.3 (B) is in EXPTIME.*

Proof. Computing the winning states and $\mathcal{H}$ on turn-based games can be done in linear time in the size of the game. As H has size exponential in the size of M and Σ_a, the algorithm we provide to solve Problem 6.3 is in EXPTIME. $\square$

We do not investigate in this thesis whether Problem 6.3 is EXPTIME-complete. This question remains open.

Example 6.4 *To illustrate this section, we consider the following small example. The system is depicted by the LTS in Figure 6.5(a) with $\Sigma = \Sigma_a = \Sigma_v = \{a, b\}$. The secret predicate ϕ is defined by $F(\phi) = \{2\}$. Figure 6.5(b) represents the associated game automaton. The states of Player 1 are represented by circles whereas the ones of Player 2 are represented by squares. The only state of Bad is the state $(\{2\}, \{a, b\})$ (bottom left) as*



(a): the automaton M

(c): an observer

(b): The game automaton

Figure 6.5: Example of a game automaton

$reach_M(\emptyset)(\{2\}) = \{2\} \subseteq F(\phi)$. The set of winning strategies of Player 1 is obtained when Player 1 does not play $\{a, b\}$ in state $(\{2\})$. This corresponds effectively to what we can observe on the LTS; if the attacker knows that the system has reached the state $\{2\}$ after an observed trace such that both a and b are observable next, then he knows for sure that the system is at state $\{2\}$ and the secret is disclosed.

To obtain the game LTS $\mathcal{H}$, it is then sufficient to remove the state $(\{2\}, \{a, b\})$ and therefore the move of Player 1 $\{a, b\}$ at state $(\{2\})$.

Note, as it is patent from M, if the observability choice consists in hiding either a or b at the beginning (i.e. for the trace ϵ), then the attacker will never be able to infer from the subsequent observed traces that the system is at state (1) or (2). An example of such an observer encoding a valid observability choice is depicted in figure 6.5(c).

6.4 Conclusion

In this chapter, we have investigated the synthesis of opaque systems when the observability of events can change over time. In the context of static observers, where the observability of events is fixed a priori, we provided an algorithm to compute a maximal subalphabet of observable events ensuring opacity. We also show that this problem is PSPACE-complete.

Then, we considered the case where the observability of events can be modified at runtime. We defined a model of dynamic projection based on the notion of observability choice, determining whether an event is observable after a given observed trace, and implemented by mean of observers which is a classical LTS augmented with observability information. We formulated the notion of valid observability choice, i.e which induces a dynamic projection enforcing opacity. We proved that verifying the validity of a regular observability choice, i.e. encoded by a finite state observer, is PSPACE-complete. We provided a method to compute the set of all valid observability choices by computing the set of winning strategies in a turn-based safety two-player game and demonstrated in that case that the set of all valid observability choices can be finitely represented. We have shown that this representation can be computed in exponential time w.r.t the size of the system.

In this chapter, we assumed that the observers can change the set of observable events only after an observable event has occurred. It would be interesting to investigate also the case where this decision depends on the word executed by the system. The case where the observability choices depend on the state of the system should also be considered.

Finally, we only gave a finite representation of all the dynamic projections enforcing opacity. A natural continuation of this work would be to search for a dynamic projection enforcing opacity that is optimal with respect to some criteria. For example, in [CDM09a, CDM09b], we proposed a solution which consists in defining cost functions over the number of observable events that need to be hidden. This permitted to select an optimal dynamic projection minimizing this cost. But is is not clear that such cost functions correspond to some optimization requirement that we can encounter for practical applications. In order to define a meaningful order relation over this set of dynamic projections, it would be interesting to investigate more availability requirements. Based on our game representation, an objective would be to select a dynamic projection ensuring the best quality of service, for example minimizing the number of unanswered requests.

7 Conclusion

In this thesis, we investigate two different kinds of problems related to the notion of opacity.

The first one is to verify whether a given system satisfies opacity. We prove that this problem is PSPACE-complete for finite systems. For infinite systems, we consider provided an abstract interpretation framework based on Galois connections. In this context, we present how to derive sound monitors allowing an attacker to infer secret information at runtime. We also show how to combine overapproximations and underapproximations to detect confidentiality vulnerabilities on a possibly infinite system. The second approach is to certify opacity on an infinite system when a regular abstraction of its language is provided. In that case, we present sufficient conditions regarding this abstraction to certify at runtime that no violation of opacity (i.e. no flow of secret information) has occurred.

The second objective is opacity enforcement and we present two approaches to solve this problem. The first one consists in restricting the system in order to confine its behavior to a secure subset. This is achieved by applying the supervisory control theory and computing a most permissive finite controller which, implemented in parallel with the system, enforces opacity. The second one consists in modifying the observability of the events in order to confuse the attacker and to prevent him from inferring the truth of a secret predicate. We have shown that this problem can be reduced to the computation of winning strategies in a safety 2-player game.

The work presented in this thesis suggest several other problems that should be interesting to investigate.

First, it would be interesting to implement the analysis techniques presented in chapter 4 to study their applicability to real systems. A possible direction for such an implementation can be to use the abstract domains library APRON [JM09].

Second, we can remark that in the game presented in Chapter 6, the computation of winning strategies, and then dynamic projections, is mostly based on the operators $post_M$ and $reach_M$. In chapter 4, we present an method to compute sound monitors that is based on approximating theses operators $post_M$ and $reach_M$. Therefore, it would be interesting to consider the problem of computing dynamic projections in the case of infinite systems, provided as in Section 4.3 an abstraction interpretation framework to overapproximate and underapproximate the state estimates of the attacker.

Third, as already suggested in the presentation of the reduction to a game in Chapter 6, the proposed game reduction technique relies on the assumption that the observability choice depends on the traces observed by the attacker. It would be interesting to study other situations where the observability depends for example on the words executed by the system. Also, in aspect oriented programing, the purpose it is improve the modularity of different aspects of a program, e.g. functionality, performance, security, etc. These aspects, e.g. logging strategies, timeout periods, are defined independently and later combined into a single program. This combination is based on *joint points* that are special generic instruction verifying if some properties are satisfied in order to proceed. Therefore, it would interesting to also investigate observability choices that depend on the last state of a run and connect the resulting techniques with the join points of aspect oriented programming.

Fourth, in Chapter 5, we provide a solution to the opacity control with the assumption that the set of events observable by the controller, Σ_o, and the ones observable by the attacker, Σ_a, are comparable. But there can be practical situations where this two sets are not comparable. It is therefore essential, in order to provide a complete theory for the opacity control problem, to also provide a solution to this general case where Σ_o and Σ_a are not comparable.

Finally, the most interesting extension of this thesis would be to investigate the opacity enforcement techniques presented in Chapters 5 and 6 in the context of probabilistic models. Indeed, with the notion of opacity, the attacker cannot infer that a secret predicate is satisfied on the basis of an observation if there exists at least one run explaining this observation which is not satisfying this predicate. But it may happen that the set of runs compatible with an observation and not satisfying a secret predicate may have a very low probability to have been executed. It that case, the attacker can for example infer the truth of a secret predicate with a very low probability of error. Therefore, the system cannot be considered as secure, even if the opacity property is not violated. In this direction, the authors of [LM05] extend the notion of opacity to the case of probabilistic system. Another approach that seems promising to treat this problem is to apply the methods developed in [CPP08, BCP08] which consists in considering the system as an information channel between the secret predicates and the attacker. A possible solution, opposite to the classical concern of information theory, can then be to decrease at most as possible the quality of this channel.

List of Figures

Bibliography

[Aba98] Martín Abadi. Protection in programming-language translations. In *In Proceedings of the 25th International Colloquium on Automata, Languages and Programming*, page 868–883. Springer-Verlag, 1998.

[AČC07] Rajeev Alur, Pavol Černý, and Swarat Chaudhuri. Model checking on trees with path equivalences. In *TACAS 2007*, page 664–678. Springer, 2007.

[AČZ06] Rajeev Alur, Pavol Černý, and Steve Zdancewic. Preserving secrecy under refinement. In *ICALP '06: Proceedings (Part II) of the 33rd International Colloquium on Automata, Languages and Programming*, page 107–118. Springer, 2006.

[AG99] Martin Abadi and Andrew D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Information and Computation*, 148:36–47, 1999.

[BAF05] Bruno Blanchet, Martín Abadi, and Cédric Fournet. Automated Verification of Selected Equivalences for Security Protocols. In *20th IEEE Symposium on Logic in Computer Science (LICS 2005)*, page 331–340, Chicago, IL, June 2005. IEEE Computer Society.

[BBB$^+$06] Eric Badouel, Marek A. Bednarczyk, Andrzej M. Borzyszkowski, Benoît Caillaud, and Philippe Darondeau. Concurrent secrets. In S. Lafortune, F. Lin, and D. Tilbury, editors, *8th Workshop on Discrete Event Systems, WODES'06*, Ann Arbor, Michigan, USA, July 2006.

[BBB$^+$07] Eric Badouel, Marek A. Bednarczyk, Andrzej M. Borzyszkowski, Benoît Caillaud, and Philippe Darondeau. Concurrent secrets. *Discrete Event Dynamic Systems*, 17:425–446, December 2007.

[BCLR09] Gilles Benattar, Franck Cassez, Didier Lime, and Olivier H. Roux. Synthesis of Non-Interferent Timed Systems. In *Proc. of the 7th Int. Conf. on Formal Modeling and Analysis of Timed Systems (FORMATS'09)*, Lecture Notes in Computer Science, Budapest, Hungary, September 2009. Copyright Springer.

[BCP08] Christelle Braun, Konstantinos Chatzikokolakis, and Catuscia Palamidessi. Compositional methods for information-hiding. In *FOSSACS'08*, Lecture Notes in Computer Science 4962, page 443–457. Springer, 2008.

[Bis04] Matt Bishop. *Introduction to computer security.* Addison-Wesley Professional, 2004.

[BKMR05] Jeremy Bryans, Maciej Koutny, Laurent Mazaré, and Peter Y. A. Ryan. Opacity generalised to transition systems. In Theo Dimitrakos, Fabio Martinelli, Peter Y. A. Ryan, and Steve A. Schneider, editors, *Revised Selected Papers of the 3rd International Workshop on Formal Aspects in Security and Trust (FAST'05)*, volume 3866 of *Lecture Notes in Computer Science*, page 81–95, Newcastle upon Tyne, UK, 2005. Springer.

[BKMR08] Jeremy Bryans, Maciej Koutny, Laurent Mazaré, and Peter Y. A. Ryan. Opacity generalised to transition systems. *International Journal of Information Security*, 7(6):421–435, 2008.

[BL73] David D.E. Bell and Leonard J. La Padula. Secure computer system: Mathematical foundations. Technical Report 2547, MITRE, March 1973.

[BS81] Stanley Burris and H. P. Sankappanavar. *A Course in Universal Algebra.* Springer-Verlag Graduate Texts in Mathematics, the millennium edition edition, 1981. Link to PDF.

[CC77a] Patrick Cousot and Radhia Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the Fourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, page 238–252, Los Angeles, California, 1977. ACM Press, New York, NY.

[CC77b] Patrick Cousot and Radhia Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *4th ACM Symposium on Principles of Programming Languages, POPL'77*, Los Angeles, January 1977.

[CC92a] Patrick Cousot and Radhia Cousot. Abstract interpretation frameworks. *Journal of Logic and Computation*, 2(4):511–547, August 1992.

[CC92b] Patrick Cousot and Radhia Cousot. Comparing the Galois connection and widening/narrowing approaches to abstract interpretation, invited paper. In M. Bruynooghe and M. Wirsing, editors, *Proceedings of the International*

Workshop Programming Language Implementation and Logic Programming, PLILP '92,, Leuven, Belgium, 13–17 August 1992, Lecture Notes in Computer Science 631, page 269–295. Springer-Verlag, Berlin, Germany, 1992.

[CDM09a] Franck Cassez, Jérémy Dubreil, and Hervé Marchand. Dynamic observers for the synthesis of opaque systems. In *7th International Symposium on Automated Technology for Verification and Analysis (ATVA'09)*, Macao SAR, China, October 2009.

[CDM09b] Franck Cassez, Jérémy Dubreil, and Hervé Marchand. Dynamic observers for the synthesis of opaque systems. Technical Report 1930, IRISA, May 2009. An extended version of the paper of ATVA'09.

[CL08] Christos G. Cassandras and Stéphane Lafortune. *Introduction to Discrete Event Systems.* Springer, 2008.

[CLRS01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms.* The MIT Press, 2nd revised edition edition, September 2001.

[CMR07a] Franck Cassez, John Mullins, and Olivier H. Roux. Synthesis of non-interferent distributed systems. In *4th Int. Conf. on Mathematical Methods, Models and Architectures for Computer Network Security (MMM-ACNS'07)*, January 2007.

[CMR07b] Franck Cassez, John Mullins, and Olivier H. Roux. Synthesis of non-interferent systems. In *4th Int. Conf. on Mathematical Methods, Models and Architectures for Computer Network Security (MMM-ACNS'07)*, volume 1 of *Lecture Notes in Computer Science Series: Communications in Computer and Inform. Science*, page 307–321. Copyright Springer, September 2007.

[CPP08] Konstantinos Chatzikokolakis, Catuscia Palamidessi, and Prakash Panangaden. Anonymity protocols as noisy channels. *Inf. Comput.*, 206(2-4):378–401, 2008.

[CT08] Franck Cassez and Stavros Tripakis. Fault diagnosis with static or dynamic diagnosers. *Fundamenta Informaticae*, 88:497–540, 2008.

[DDHR06] Martin De Wulf, Laurent Doyen, Thomas A. Henzinger, and Jean-François Raskin. Antichains: A new algorithm for checking universality of finite automata. In *Proceedings of CAV 2006: Computer-Aided Verification*, Lecture Notes in Computer Science 4144, page 17–30. Springer-Verlag, 2006.

[DDM08] Jérémy Dubreil, Philippe Darondeau, and Hervé Marchand. Opacity Enforcing Control Synthesis. In B. Lennartson, M. Fabian, K. Akesson, A. Giua, and R. Kumar, editors, *Proceedings of the 9th International Workshop on Discrete Event Systems (WODES'08)*, page 28–35, Göteborg, Sweden, May 2008. IEEE.

[DDM09] Jérémy Dubreil, Philippe Darondeau, and Hervé Marchand. Supervisory control for opacity. *IEEE Transactions on Automatic Control*, 2009. To appear.

[DDMR08a] Martin De Wulf, Laurent Doyen, Nicolas Maquet, and Jean-François Raskin. Alaska: Antichains for logic, automata and symbolic kripke structures analysis. In *ATVA: Automated Technology for Verification and Analysis*, Lecture Notes in Computer Science. Springer-Verlag, 2008.

[DDMR08b] Martin De Wulf, Laurent Doyen, Nicolas Maquet, and Jean-François Raskin. Antichains: Alternative algorithms for ltl satisfiability and model-checking. In *TACAS: Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science 4963, page 63–77. Springer-Verlag, 2008.

[DDR06] Martin De Wulf, Laurent Doyen, and Jean-François Raskin. A lattice theory for solving games of imperfect information. In *Proceedings of HSCC 2006: Hybrid Systems—Computation and Control*, Lecture Notes in Computer Science 3927, page 153–168. Springer-Verlag, 2006.

[DFG+06] Vianney Darmaillacq, Jean-Claude Fernandez, Roland Groz, Laurent Mounier, and Jean-Luc Richier. Test generation for network security rules. In *TestCom 2006*, volume 3964 of *LNCS*, 2006.

[DGMD06] Mila Dalla Preda, Roberto Giacobazzi, Matias Madou, and Koen De Bosschere. Opaque predicates detection by abstract interpretation. In *Proc. of the 11th International Conference on Algebraic Methodology and Software Technology* (*AMAST '06*), volume 4019 of *Lecture Notes in Computer Science*, page 81–95. Springer-Verlag, 2006.

[DJM07] Jérémy Dubreil, Thierry Jéron, and Hervé Marchand. Construction de moniteurs pour la surveillance de propriétés de sécurité. In *6ème Colloque Francophone sur la Modélisation des Systèmes Réactifs (MSR'07)*, Lyon, France, October 2007.

[DJM09] Jérémy Dubreil, Thierry Jéron, and Hervé Marchand. Monitoring confidentiality by diagnosis techniques. In *Proceedings of the 10th European Control Conference (ECC'09)*, Budapest, Hungary, August 2009.

[DLT00] Rami Debouk, Stéphane Lafortune, and Demosthenis Teneketzis. Coordinated decentralized protocols for failure diagnosis of discrete event systems. *Discrete Event Dynamic Systems*, 10(1-2):33–86, 2000.

[Dub09] Jérémy Dubreil. Opacity and abstraction. In *Proceedings of the First International Workshop on Abstractions for Petri Nets and Other Models of Concurrency (APNOC'09)*, Paris, France, June 2009.

[FFM09] Yliès Falcone, Jean-Claude Fernandez, and Laurent Mounier. Enforcement monitoring wrt. the safety-progress classification of properties. In *SAC*, page 593–600, 2009.

[FG93] Riccardo Focardi and Roberto Gorrieri. An information flow security property for CCS. In *Proceedings of Second North American Process Algebra Workshop*, 1993.

[FG00] Riccardo Focardi and Roberto Gorrieri. Classification of security properties: Information flow. In *Foundations of Security Analysis and Design, LNCS No 2171*, page 331–396, 2000.

[FG01] Riccardo Focardi and Roberto Gorrieri. Classification of security properties (part I: Information flow). In Riccardo Focardi and Roberto Gorrieri, editors, *Foundations of Security Analysis and Design I: FOSAD 2000 Tutorial Lectures*, volume 2171 of *Lecture Notes in Computer Science*, page 331–396, Heidelberg, 2001. Springer-Verlag.

[GM82] Joseph A. Goguen and José Meseguer. Security policies and security models. In *1982 Berkeley Conference on Computer Security*, page 11–20. IEEE Computer Society, April 1982.

[GM04] Roberto Giacobazzi and Isabella Mastroeni. Abstract non-interference: parameterizing non-interference by abstract interpretation. In *POPL '04: Proceedings of the 31st ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, page 186–197. ACM, 2004.

[GMP92] Janice I. Glasgow, Glenn H. MacEwen, and Prakash Panangaden. A logic for reasoning about security. *ACM Transactions on Computer Systems*, 10(3):226–264, 1992.

[Hin62] Jaakko Hintikka. *Knowledge and Belief.* Cornell university Press, 1962.

[JM09] Bertrand Jeannet and Antoine Miné. Apron: A library of numerical abstract domains for static analysis. In *Proceedings of CAV 2009: Computer-Aided Verification*, Lecture Notes in Computer Science 5643, page 661–667. Springer, 2009.

[JMGL08] Thierry Jéron, Hervé Marchand, Sahika Genc, and Stéphane Lafortune. Predictability of sequence patterns in discrete event systems. In *IFAC World Congress*, Seoul, Korea, July 2008.

[JMPC06] Thierry Jéron, Hervé Marchand, Sophie Pinchinat, and Marie-Odile Cordier. Supervision patterns in discrete event systems diagnosis. In *Workshop on Discrete Event Systems, WODES'06*, Ann-Arbor (MI, USA), July 2006.

[KJ04] Ratnesh Kumar and Shengbing Jiang. Failure diagnosis of discrete event systems with linear-time temporal logic specifications. *IEEE Transactions on Automatic Control*, 49(6):934–945, 2004.

[Lam77] L. Lamport. Proving the correctness of multiprocess programs. *IEEE Trans. Softw. Eng.*, 3(2):125–143, 1977.

[LBW05] Jay Ligatti, Lujo Bauer, and David Walker. Edit automata: Enforcement mechanisms for run-time security policies. *International Journal of Information Security*, 4(1–2):2–16, February 2005.

[LJ07] Tristan Le Gall and Bertrand Jeannet. Lattice automata: a representation of languages over an infinite alphabet, and some applications to verification. In *The 14th International Static Analysis Symposium, SAS 2007*, number 4634 in LNCS, page 52–68, Kongens Lyngby, Denmark, August 2007.

[LJJ06] Tristan Le Gall, Bertrand Jeannet, and Thierry Jéron. Verification of communication protocols using abstract interpretation of fifo queues. In Michael Johnson and Varmo Vene, editors, *11th International Conference on Algebraic Methodology and Software Technology, AMAST '06, Kuressaare, Estonia*, volume 4019 of *LNCS*, page 204–219. Springer-Verlag, July 2006.

[LM05] Yassine Lakhnech and Laurent Mazaré. Probabilistic opacity for a passive adversary and its application to chaum's voting scheme. Technical Report TR-2005-4, Verimag, Centre Équation, 38610 Gières, February 2005.

[Low99] Gavin Lowe. Towards a completeness result for model checking of security protocols. *Journal of Computer Security*, 7(2-3):89–146, 1999.

[Low02] Gavin Lowe. Quantifying information flow. In *15th IEEE Computer Security Foundations Workshop (CSFW-15 2002), 24-26 June 2002, Cape Breton, Nova Scotia, Canada*, pages 18–31. IEEE Computer Society, 2002.

[Mar75] Donald A. Martin. Borel determinacy. *Annals of Mathematics*, 102(2):363–371, 1975.

[Mas05] Isabella Mastroeni. *Abstract Non-Interference: An Abstract Interpretation-based approach to Secure Information Flow.* PhD thesis, Università di Verona, Italy, March 2005.

[Mas08] Isabella Mastroeni. Deriving bisimulations by simplifying partitions. In *Nineth International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI'08)*, volume 4905 of *Lecture Notes in Computer Science*, page 157–171. Springer-Verlag, 2008.

[Maz04] Laurent Mazaré. Using unification for opacity properties. In *Proceedings of the 4th IFIP WG1.7 Workshop on Issues in the Theory of Security (WITS'04)*, page 165–176, Barcelona (Spain), 2004.

[Mcl94] John Mclean. A general theory of composition for trace sets closed under selective interleaving functions. In *In Proc. IEEE Symposium on Security and Privacy*, page 79–93, 1994.

[MDJ09] Hervé Marchand, Jérémy Dubreil, and Thierry Jéron. Automatic testing of access control for security properties. In *TestCom'09*, 2009.

[MVO96] Alfred J. Menezes, Scott A. Vanstone, and Paul C. Van Oorschot. *Handbook of Applied Cryptography.* CRC Press, Inc., Boca Raton, FL, USA, 1996.

[Ner58] Anil Nerode. Linear automaton transformations. *Proc. AMS*, 9:541–544, 1958.

[QK04] Wenbin Qiu and Ratnesh Kumar. Decentralized failure diagnosis of discrete event systems. In *WODES'04*, 2004.

[Ric06] S. Laurie Ricker. A question of access: decentralized control and communication strategies for security policies. In *8th International Workshop on Discrete Event Systems*, page 58 – 63, June 2006.

[RS99] Peter Y.A. Ryan and Steve A. Schneider. Process algebra and non-interference. In *Journal of Computer Security*, page 214–227, 1999.

[RW87] Peter J. Ramadge and W. Murray Wonham. Supervisory control of a class of discrete event processes. *SIAM J. Control Optim.*, 25(1):206–230, January 1987.

[RW89] Peter J. Ramadge and W. Murray Wonham. The control of discrete event systems. *Proceedings of the IEEE; Special issue on Dynamics of Discrete Event Systems*, 77(1):81–98, 1989.

[Sch00] Fred B. Schneider. Enforceable security policies. *ACM Trans. Inf. Syst. Secur.*, 3(1):30–50, 2000.

[SH08] Anooshiravan Saboori and Christoforos N. Hadjicostis. Verification of the Initial-State opacity in Security Applications of DES. In B. Lennartson, M. Fabian, K. Akesson, A. Giua, and R. Kumar, editors, *Proceedings of the 9th International Workshop on Discrete Event Systems (WODES'08)*, Göteborg, Sweden, May 2008. IEEE.

[SLS$^+$96] Mera Sampath, Stéphane Lafortune, Kasim Sinaamohideen, Demosthenis Teneketzis, and Raja Sengupta. Failure diagnosis using discrete event models. *IEEE Transactions on Control System Technology*, 1996.

[SM73] Larry J. Stockmeyer and Albert R. Meyer. Word problems requiring exponential time: Preliminary report. In *STOC*, page 1–9, 1973.

[SM03] Andrei Sabelfeld and Andrew C. Myers. Language-based information-flow security. *IEEE Journal on Selected Areas in Communications*, 21:2003, 2003.

[SS96] Steve A. Schneider and Abraham Sidiropoulos. Csp and anonymity. In *In European Symposium on Research in Computer Security*, page 198–218. Springer-Verlag, 1996.

[SSL$^+$95] Mera Sampath, Raja Sengupta, Stéphane Lafortune, Kasim Sinaamohideen, and Demosthenis Teneketzis. Diagnosability of discrete event systems. *IEEE Transactions on Automatic Control*, 1995.

[Tar55] Alfred Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.

[Tho95] Wolfgang Thomas. On the synthesis of strategies in infinite games. In *Proc. 12th Annual Symposium on Theoretical Aspects of Computer Science (STACS'95)*, volume 900, page 1–13. Springer, 1995. Invited talk.

[TO08] Shigemasa Takai and Yusuke Oka. A formula for the supremal controllable and opaque sublanguage arising in supervisory control. *SICE Journal of Control, Measurement, and System Integration*, 1(4):307–312, March 2008.