Introduction au Model Checking

Université de Nantes
Département Informatique

2024 – 2025

Master 2 Informatique
Architectures Logicielles (ALMA)

Guillaume Cantin & Guilhem Jaber

Table des matières

Introduction

Si l'histoire récente du progrès technologique est marquée par de très nombreux succès retentissants, notamment dans le domaine de la conquête spatiale et des réseaux de communication, elle est aussi jalonnée de célèbres échecs. Ainsi, le premier lancement de la fusée Ariane V, le 4 juin 1996, depuis la base de Kourou en Guyane française, a été marqué par une explosion fatale, 36 secondes après le décollage. Une commission d'expertise, dirigée par le mathématicien français Jacques-Louis Lions, a conclu que cette explosion a été provoquée par un défaut du système informatique de pilotage de la fusée : un dépassement d'entier dans les registres mémoire des calculateurs électroniques utilisés par le pilote automatique aurait en effet provoqué la panne du système de navigation de la fusée, causant sa destruction. Si cette explosion n'a causé que des dégâts matériels, elle a néanmoins conduit à une perte financière très importante. On pourrait multiplier les exemples de catastrophes technologiques dues à ce type d'erreur : l'affaire de la machine de radio-thérapie Therac 25 entre 1985 et 1987, la défaillance du système antimissile Patriot en 1991, le dysfonctionnement du microprocesseur Pentium en 1994, etc. La présence d'erreurs dans le fonctionnement des systèmes utilisés constitue le point commun de ces différents échecs.

La *vérification* du bon fonctionnement des systèmes qui nous entourent représente donc un enjeu majeur, pour de très nombreux secteurs de nos sociétés, comme les transports, la santé, l'informatique ou l'industrie. La nécessaire vérification constitue aujourd'hui une étape longue et coûteuse du processus de développement de nouveaux outils. L'élaboration de méthodes de vérification rigoureuses et sûres, permettant de réduire au maximum le risque d'erreur, constitue un domaine de recherche scientifique en informatique, qui s'appuie fortement sur la logique mathématique et les méthodes formelles. Ce cours constitue une introduction à quelques techniques de vérification formelle de systèmes, qui reposent sur la vérification de *modèles* de ces systèmes, méthode plus connue sous son nom anglais *Model Checking*.

Naissance et fonctionnement du Model Checking

Le Model Checking est né au début des années 1980, quasi-simultanément en deux endroits : en France, à Grenoble, avec Queille et Sifakis, qui ont développé le système CESAR et sa logique temporelle [15], et aux USA avec Clarke et Emerson qui ont développé la logique temporelle CTL (*Computation Tree Logic*). Ces travaux ont donné le prix Turing 2007 à Clarke, Emerson et Sifakis. Ils s'appuyaient eux-mêmes sur les travaux de Pnueli (prix Turing en 1996) sur la logique temporelle. Le Model Checking s'est considérablement développé ensuite, et constitue certainement la méthode formelle la plus utilisée dans l'industrie, en particulier dans la CAO (Conception Assistée par Ordinateur) de circuits.

Le principe de base du Model Checking est que la vérification des propriétés d'un système est

réalisée non pas sur le système lui-même, mais sur un modèle de ce système. La modélisation des systèmes constitue donc une étape cruciale des méthodes de vérification. L'utilisation des structures de Kripke ou des systèmes de transition est très largement répandue, notamment depuis les travaux de Milner (prix Turing 1992) sur l'étude des processus dits *parallèles*. Puis, la spécification des propriétés à vérifier, dans un certain langage formel, permet l'automatisation du processus de vérification, comme illustré sur la figure 1. L'automatisation du Model Checking repose ensuite sur le développement d'algorithmes, prenant en entrée un modèle de système et une propriété formalisée dans un certain système logique, et décidant si oui ou non le système vérifie la propriété considérée. Dans le cas ou la propriété n'est pas satisfaite, on souhaite de plus compléter la réponse de l'algorithme en donnant un contre-exemple sous la forme d'une exécution du modèle, permettant de localiser précisément le point où l'erreur se produit. Les modèles utilisés admettent très souvent un nombre raisonnable d'états et de transitions, afin de garantir une complexité algorithmique réalisable. Récemment, le développement de systèmes de transition hybrides, associant des modèles discrets et des modèles continus, ouvre une nouvelle perspective de recherche pour la vérification de systèmes dont le comportement est difficile à modéliser, notamment pour des systèmes issus des sciences du vivant.

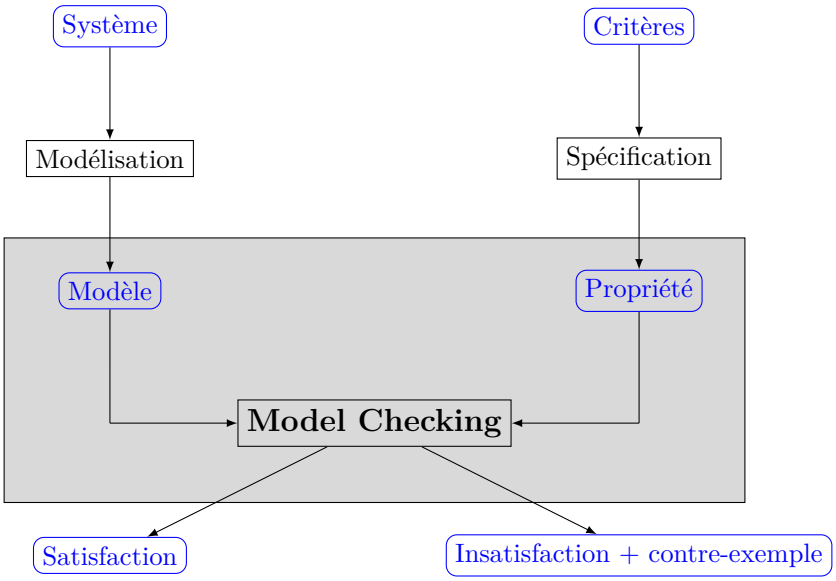

Figure 1. *Description schématique du processus de Model Checking.*

Un exemple simple de système technologique omniprésent dans notre entourage, et dont on souhaite vivement assurer le bon fonctionnement, est celui de l'ascenseur. On peut facilement modéliser le fonctionnement d'un ascenseur par un système de transition admettant un nombre fini d'états, puis exprimer des propriétés dites de *sûreté*, comme « *à aucun moment l'ascenseur ne peut voyager la porte ouverte* », d'absence de blocage de l'exécution, ou de *progrès*, comme « *l'ascenseur finira par répondre à toutes les demandes des passagers* ». Les méthodes du Model Checking permettent alors une vérification de ces propriétés essentielles.

Il est important de noter que la validité du processus de vérification, qui peut lui-même contenir des erreurs, produit une réponse sur le modèle du système et non pas sur le système lui-même. La qualité et l'efficacité du processus de vérification sont donc limitées par celles de l'effort de modélisation qui le précède.

Plan du cours

Ce cours d'introduction au Model Checking est divisé en quatre chapitres. Le premier chapitre est consacré à la modélisation de systèmes réels par des systèmes de transition. On y présente le matériel

de base nécessaire à la mise en œuvre à venir des algorithmes de Model Checking, avec la notion d'exécution d'un système de transition, et les opérations de composition de tels systèmes, permettant notamment de modéliser le comportement de systèmes concurrents.

Le deuxième chapitre concerne les propriétés linéaires, qui sont une classe importante de propriétés permettant de décrire le comportement de systèmes de transition. On introduit la notion de *trace* d'un système de transition et on construit une première méthode algorithmique de vérification de propriétés dites *invariantes*, qui décrivent certains comportements d'un système de transition, et sur lesquelles s'appuie la vérification d'autres propriétés. Quelques rappels essentiels sur les graphes et les langages sont proposés, pour assurer un niveau d'autosuffisance raisonnable à ce document.

Dans le troisième chapitre, on présente une méthode algorithmique de vérification d'une classe plus importante de propriétés linéaires, appelées *propriétés de sûreté*. On s'intéresse d'abord aux propriétés de sûreté dites *régulières*, qui peuvent être reconnues par des automates finis. La complexité du problème de décision correspondant à la vérification de telles propriétés est établie. Le chapitre se termine par une ouverture vers la vérification d'autres types de propriétés.

Enfin, le quatrième chapitre est consacré à la logique temporelle linéaire, un système logique introduit par Amir Pnueli en 1977 pour la vérification de programmes informatiques, ainsi qu'à une extension de cette logique, adaptée à la vérification de modèles non nécessairement discrets. Une méthode élémentaire de vérification statistique est finalement présentée, pour montrer un aperçu des avancées très récentes dans ce domaine de recherche scientifique.

CHAPITRE 1

Modèles de systèmes

Lorsque l'on souhaite étudier les propriétés d'un système réel, une étape préliminaire au processus de vérification consiste à élaborer un *modèle* de ce système. Les méthodes de vérification mises en œuvre à la suite de cette phase de modélisation, aussi robustes soient-elles, apportent une réponse sur la validité des propriétés qui correspond alors au modèle du système, et non pas au système lui-même. Il est donc primordial de construire un modèle qui puisse décrire aussi fidèlement que possible le comportement du système que l'on souhaite analyser. Nous allons découvrir dans ce chapitre une classe importante de modèles, utilisés dans différents contextes : les systèmes de transition.

1.1 Systèmes de transition

Les systèmes de transition sont souvent utilisés comme *modèles* pour étudier le comportement de systèmes, issus notamment de l'informatique, de l'ingénierie, de l'industrie, mais aussi des sciences du vivant.

Définition 1.1 (Système de transition)**.** Un *système de transition* ST est déterminé par un sextuplet $(S, Act, \to, I, Prop, L)$ dans lequel :

- S est un ensemble d'*états*,
- Act est un ensemble d'*actions*,
- $\to \subseteq S \times Act \times S$ est une *relation de transition*,
- $I \subseteq S$ est un ensemble d'*états initiaux*,
- $Prop$ est un ensemble de *propositions atomiques*,
- $L : S \longrightarrow 2^{Prop}$ est une *fonction d'étiquetage*.

Un système de transtion (ST) est dit *fini* si les ensembles S, Act et $Prop$ sont finis. □

Si $(s, \alpha, s') \in \to$, on écrit généralement $s \xrightarrow{\alpha} s'$.

Rappel. Si X est un ensemble, la notation 2^X désigne l'ensemble des parties de X. Une autre notation fréquemment utilisée est $\mathscr{P}(X)$. ○

Remarque. Les systèmes de transition sont mal nommés, car ce sont des *modèles* de systèmes ! ◁

Exemple 1.1 (L'ascenseur)**.** On considère un ascenseur qui dessert les trois niveaux N_0, N_1 et N_2 d'un bâtiment. Cet ascenseur est modélisé par un système de transition illustré sur la figure 1.1. Les

états de ce système correspondent aux trois niveaux du bâtiment :

$$S = \{N_0, N_1, N_2\}.$$

L'ensemble des actions est :

$$Act = \{\mathsf{m\_vide}, \mathsf{m\_charge}, \mathsf{d\_vide}, \mathsf{d\_charge}\}.$$

Ces actions correspondent aux mouvements de montée ou de descente de l'ascenseur, qui peut effectuer des déplacements en étant vide ou en étant chargé. La relation de transition $\rightarrow$ est composée des transitions suivantes :

$$\begin{array}{ll} (N_0, \mathsf{m\_vide}, N_1), & (N_0, \mathsf{m\_charge}, N_1), \\ (N_1, \mathsf{m\_vide}, N_2), & (N_1, \mathsf{m\_charge}, N_2), \\ (N_2, \mathsf{d\_vide}, N_1), & (N_2, \mathsf{d\_charge}, N_1), \\ (N_1, \mathsf{d\_vide}, N_0), & (N_1, \mathsf{d\_charge}, N_0). \end{array}$$

L'ensemble des états initiaux est arbitraire. On peut considérer par exemple

$$I = \{N_0\}.$$

L'ensemble des propositions atomiques est lui aussi arbitraire. On peut supposer par exemple qu'il est donné par :

$$Prop = \{o, f, N_0, N_1, N_2\},$$

où les propositions o et f modélisent l'ascenseur avec portes ouvertes et portes fermées respectivement. Enfin, la fonction d'étiquetage est définie par :

$$\begin{aligned} L(N_0) &= \{o, f, N_0\}, \\ L(N_1) &= \{o, f, N_1\}, \\ L(N_2) &= \{o, f, N_2\}. \end{aligned}$$

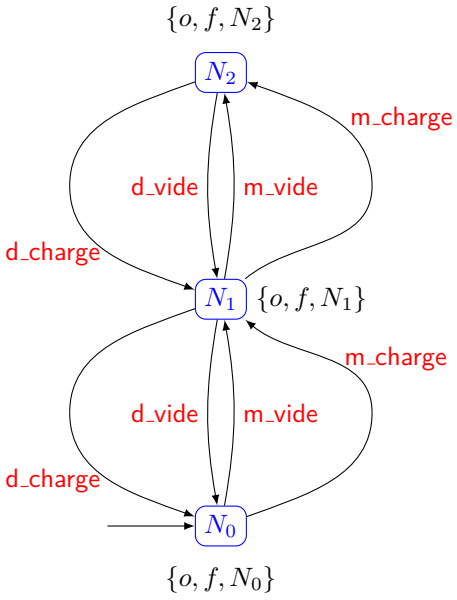

Figure 1.1. *Système de transition modélisant le fonctionnement d'un ascenseur.*

•

Remarque. L'exemple de l'ascenseur est l'exemple de référence dans le cours d'introduction sur le *Model Checking* au Collège de France de Gérard Berry, accessible par le lien suivant.

https://www.college-de-france.fr/site/gerard-berry/course-2015-03-25-16h00.htm

◁

Rappel (Logique propositionnelle)**.** Soit P un ensemble de propositions. L'ensemble des formules propositionnelles logiques sur P est défini par les 4 règles suivantes :

- « Vrai » est une formule ;
- toute proposition est une formule ;
- si Φ_1, Φ_2 et Φ sont des formules, alors $\neg\Phi$ est une formule (négation de Φ) et $\Phi_1 \wedge \Phi_2$ est aussi une formule (conjonction de Φ_1 et Φ_2 : $\wedge$ signifie « *et* ») ;
- rien d'autre n'est une formule.

L'opérateur unaire de négation $\neg$ est prioritaire sur l'opérateur binaire de conjonction $\wedge$, ce qui signifie :

$$\neg a \wedge b = (\neg a) \wedge b.$$

On a également les opérateurs binaires suivants :

$$\begin{array}{ll} \Phi_1 \vee \Phi_2 = \neg(\neg\Phi_1 \wedge \neg\Phi_2) & \text{(disjonction)}, \\ \Phi_1 \Rightarrow \Phi_2 = \neg\Phi_1 \vee \Phi_2 & \text{(implication)}, \\ \Phi_1 \Leftrightarrow \Phi_2 = (\neg\Phi_1 \wedge \neg\Phi_2) \vee (\Phi_1 \wedge \Phi_2) & \text{(équivalence)}, \\ \Phi_1 \oplus \Phi_2 = (\neg\Phi_1 \wedge \Phi_2) \vee (\Phi_1 \wedge \neg\Phi_2) & \text{(parité)}. \end{array}$$

L'opérateur de parité est également appelé « ou exclusif » et il est parfois noté XOR (*exclusive or* en anglais). ○

Le comportement d'un système de transition est décrit de la façon suivante. Le système démarre dans un état initial $s_0 \in I$ puis évolue selon la relation de transition $\rightarrow$. Les transitions peuvent être sélectionnées de façon *non déterministe*. Cette situation peut se produire si un état admet plusieurs transitions de sortie.

La fonction d'étiquetage L associe à chaque état $s \in S$ un sous-ensemble $L(s) \subseteq \mathit{Prop}$. Ainsi, $L(s)$ correspond à l'ensemble des propositions $p \in \mathit{Prop}$ qui peuvent être satisfaites par s. Si Φ est une proposition logique, alors un état $s \in S$ satisfait la formule Φ si l'évaluation induite par $L(s)$ rend la formule Φ vraie, c'est-à-dire :

$$s \models \Phi \Leftrightarrow L(s) \models \Phi.$$

Exemple 1.2 (Le distributeur de boissons)**.** On considère un distributeur de boissons, représenté sur la figure 1.2. La machine délivre du café ou des sodas. Les états sont représentés par des rectangles aux coins arrondis. Le nom des états est écrit à l'intérieur des rectangles. Les transitions sont représentées par des flèches étiquetées. Les états initiaux sont indiqués par une flèche sans origine. Les propositions de ce système de transition dépendent des propriétés étudiées. On peut supposer que les propriétés intéressantes ne dépendent pas de la boisson choisie ; par exemple :

« *le distributeur ne délivre une boisson qu'après insertion d'une pièce* ».

Dans cet exemple, on définit donc $\mathit{Prop} = \{\mathsf{payé}, \mathsf{délivré}\}$.

•

Remarque. Cet exemple illustre le fait que l'ensemble *Prop* d'un système de transition peut varier en fonction des propriétés que l'on souhaite étudier. Souvent, cet ensemble n'est pas défini de façon explicite. Parfois, on considère même qu'il est donné par $\mathit{Prop} \subseteq S$ et on pose alors

$$L(s) = \{s\} \cap \mathit{Prop}.$$

◁

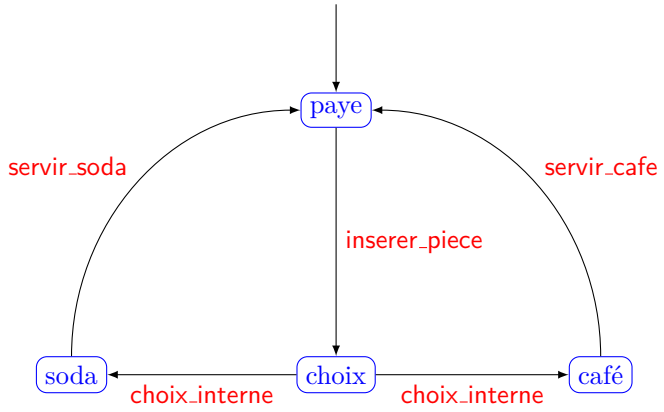

Figure 1.2. *Exemple de système de transition : un distributeur de boissons.*

Exemple 1.3 (Circuit séquentiel *hardware*)**.** On considère un circuit séquentiel avec une variable d'entrée x, une variable de sortie y et un enregistrement r. La fonction de contrôle pour la variable y est donnée par

$$\lambda_y = \neg(x \oplus r),$$

où $\oplus$ désigne le « ou » exclusif (XOR ou encore fonction de parité). L'évaluation de l'enregistrement est modifiée par la fonction de circuit

$$\delta_r = x \vee r.$$

On peut modéliser ce ciruit par un système de transition.

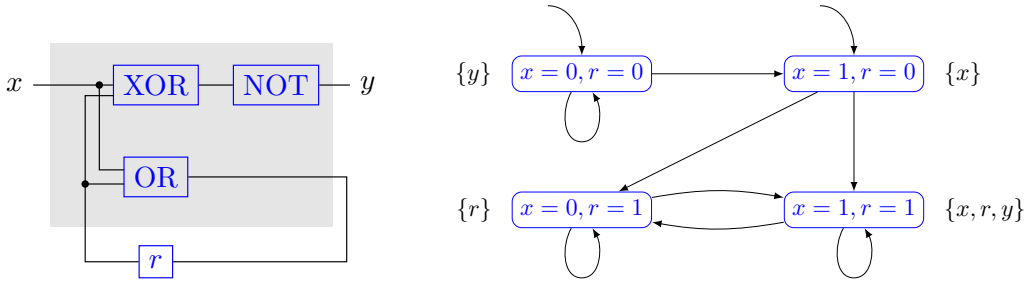

Figure 1.3. *Système de transition représentant un circuit séquentiel* hardware.

•

Définition 1.2 (Prédécesseurs, successeurs)**.** Soit $ST = (S, Act, \rightarrow, I, Prop, L)$ un système de transition. Pour $s \in S$ et $\alpha \in Act$, l'ensemble des α-successeurs directs de s est défini par

$$Post(s, \alpha) = \{s' \in S \mid s \xrightarrow{\alpha} s'\}.$$

L'ensemble des successeurs directs de s est défini par

$$Post(s) = \bigcup_{\alpha \in Act} Post(s, \alpha).$$

L'ensemble des α-prédécesseurs directs de s est défini par

$$Pre(s, \alpha) = \{s' \in S \mid s' \xrightarrow{\alpha} s\}.$$

L'ensemble des prédécesseurs directs de s est défini par

$$Pre(s) = \bigcup_{\alpha \in Act} Pre(s, \alpha).$$

Pour $C \subseteq S$, on définit par analogie les ensembles :

$$Post(C, \alpha) = \bigcup_{s \in C} Post(s, \alpha), \quad Post(C) = \bigcup_{s \in C} Post(s),$$

$$Pre(C, \alpha) = \bigcup_{s \in C} Pre(s, \alpha), \qquad Pre(C) = \bigcup_{s \in C} Pre(s).$$

□

Définition 1.3 (État final)**.** Un état s d'un système de transition ST est dit *final* (ou *terminal*) lorsque $Post(s) = \varnothing$. □

Définition 1.4 (Système de transition déterministe)**.** Soit $ST = (S, Act, \to, I, Prop, L)$ un système de transition.

On dit que ST est *déterministe par actions* si $|I| \leq 1$ et $|Post(s, \alpha)| \leq 1$ pour tout état $s \in S$ et pour toute action $\alpha \in Act$.

On dit que ST est *déterministe par propositions* si $|I| \leq 1$ et $|Post(s) \cap \{s' \in S \mid L(s') = A\}| \leq 1$ pour tout état $s \in S$ et pour tout sous-ensemble de propositions $A \subseteq Prop$. □

1.2 Structure de Kripke

Une structure proche du système de transition est donnée par la structure de Kripke. Si *Prop* désigne un ensemble de propositions atomiques, une structure de Kripke sur *Prop* est déterminée par un quadruplet $M = (S, I, R, L)$ dans lequel

- S est un ensemble fini d'états ;
- $I \subseteq S$ est un ensemble fini d'états initiaux ;
- $R \subseteq S \times S$ est une relation de transition vérifiant la propriété

$$\forall s \in S, \exists s' \in S \ ; \ (s, s') \in R;$$

- L est une fonction d'étiquetage définie sur S, à valeurs dans 2^{Prop}.

Les systèmes de transition admettent donc, en plus des structures de Kripke, un ensemble d'actions.

Selon les auteurs et les contextes, les définitions de système de transition et de structure de Kripke peuvent varier. Par exemple, dans [7], les systèmes de transition sont définis par des triplets, alors que dans [4], les structures de Kripke sont définies par des sextuplets.

1.3 Exécutions

On peut décrire formellement le comportement d'un système de transition avec la notion d'*exécution.*

Définition 1.5 (Fragment d'exécution)**.** Soit $ST = (S, Act, \to, I, Prop, L)$ un système de transition.

Un *fragment d'exécution fini* ρ de ST est une suite finie de la forme

$$\rho = s_0 \alpha_1 s_1 \alpha_2 \dots \alpha_n s_n,$$

avec $n \geq 0$ et telle que $s_i \xrightarrow{\alpha_{i+1}} s_{i+1}$ pour tout $i \in \{0, \dots, n-1\}$. On dit que n est la longueur du fragment d'exécution ρ.

Un *fragment d'exécution infini* ρ de ST est une suite infinie de la forme

$$\rho = s_0\alpha_1 s_1\alpha_2 s_2\alpha_3 s_3 \ldots,$$

telle que $s_i \xrightarrow{\alpha_{i+1}} s_{i+1}$ pour tout $i \geq 0$. □

Dans un fragment d'exécution, on a donc une alternance d'états et d'actions. Les fragments d'exécution finis sont souvent notés

$$\rho = s_0 \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_n} s_n$$

et les fragments d'exécution infinis sont souvent notés

$$\rho = s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} \ldots .$$

Définition 1.6 (Fragment d'exécution initial, maximal)**.** Un fragment d'exécution *maximal* est soit un fragment d'exécution fini qui termine dans un état final, soit un fragment d'exécution infini.

Un fragment d'exécution *initial* est un fragment d'exécution qui commence dans un état initial ($s_0 \in I$). □

Définition 1.7 (Exécution)**.** Une *exécution* d'un système de transition ST est un fragment d'exécution initial et maximal. □

Définition 1.8 (État accessible)**.** Soit $ST = (S, Act, \rightarrow, I, Prop, L)$ un système de transition. Un état $s \in S$ est dit *accessible* dans ST s'il existe un fragment d'exécution fini et initial qui termine en s. On peut donc écrire

$$s_0 \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_n} s_n,$$

avec $s_0 \in I$ et $s_n = s$.

L'ensemble des états accessibles est noté $Access(ST)$. □

1.4 Graphes conditionnels

On utilise parfois des transitions conditionnelles pour définir un système de transition. On obtient un graphe dont les arêtes sont étiquetées par des conditions. Ce graphe peut engendrer un système de transition par un processus de déploiement. Pour cela, on utilise la notion de *graphe conditionnel* (*program graph* en anglais).

On considère alors un ensemble Var de variables typées (par exemple, des entiers, des booléens ou des caractères standard). Le type d'une variable x est appelé *domaine* de x et il est noté $dom(x)$. On note ensuite $Eval(Var)$ l'ensemble des évaluations qui affectent des valeurs aux variables et $Cond(Var)$ l'ensemble des conditions booléennes sur Var.

Définition 1.9 (Graphe conditionnel)**.** Un *graphe conditionnel* GC sur un ensemble de variables typées Var est déterminé par un sextuplet $(Loc, Act, Effet, \hookrightarrow, Loc_0, g_0)$ dans lequel :

- Loc est un ensemble de lieux ;
- Act est un ensemble d'actions ;
- $Effet : Act \times Eval(Var) \longrightarrow Eval(Var)$ est une fonction décrivant l'effet d'une action sur l'évaluation des variables ;
- $\hookrightarrow \subseteq Loc \times Cond(Var) \times Act \times Loc$ est une relation de transitions conditionnelles ;
- $Loc_0 \subseteq Loc$ est un ensemble de lieux initiaux ;
- $g_0 \in Cond(Var)$ est la condition initiale.

□

On note généralement $l \xhookrightarrow{g:\alpha} l'$ à la place de $(l, g, \alpha, l') \in \hookrightarrow$. La condition g est appelée *garde* de la transition conditionnelle $l \xhookrightarrow{g:\alpha} l'$. Si le garde est une tautologie (par exemple, $g = \mathsf{Vrai}$ ou $g = (x < 1) \vee (x \geq 1)$), alors on écrit plus simplement $l \xhookrightarrow{\alpha} l'$.

Chaque graphe conditionnel GC peut être interprété comme un système de transition ST, par un processus de déploiement. Les états qui en résultent sont des couples de la forme $< l, \eta >$ avec $l \in Loc$ et $\eta \in Eval(Var)$.

Définition 1.10 (Système de transition engendré par un graphe conditionnel)**.** On considère un graphe conditionnel $GC = (Loc, Act, Effet, \hookrightarrow, Loc_0, g_0)$ sur un ensemble de variables typées Var. Le système de transition ST engendré par GC est déterminé par le sextuplet $(S, Act, \rightarrow, I, Prop, L)$ dans lequel :

- $S = Loc \times Eval(Var)$;
- l'ensemble d'actions Act est celui de GC ;
- $\rightarrow \subseteq S \times Act \times S$ est défini par la règle

$$\frac{l \xhookrightarrow{g:\alpha} l' \wedge \eta \models g}{< l, \eta > \xrightarrow{\alpha} < l', Effet(\alpha, \eta) >};$$

- $I = \{< l, \eta > \mid l \in Loc_0, \eta \models g_0\}$;
- $Prop = Loc \cup Cond(Var)$;
- $L(< l, \eta >) = \{l\} \cup \{g \in Cond(Var) \mid \eta \models g\}$.

□

La définition du système de transition ST engendré par le graphe conditionnel GC détermine *a priori* un ensemble de propositions très grand. Mais, généralement, on réduit sa taille pour décrire quelques propriétés du système.

1.5 Parallélisme

De nombreux systèmes évoluent en parallèle d'autres systèmes. Pour décrire ce type de situation, nous allons définir un opérateur $\|$ associatif et commutatif, tel que le comportement du système de transition

$$ST = ST_1 \parallel ST_2 \parallel \cdots \parallel ST_n$$

reflète la composition parallèle des systèmes de transition $ST_1, \ldots, ST_n$.

Avant de définir l'opérateur $\|$, nous allons construire un opérateur d'entrelacement $|||$ (*interleaving* en anglais). Nous verrons plus loin que l'opérateur d'entrelacement $|||$ est un cas particulier de l'opérateur de composition $\|$.

1.5.1 Entrelacement de systèmes de transition

Exemple 1.4 (Feux de circulation indépendants)**.** On considère deux feux de circulation placés sur des routes qui ne se coupent pas. Chaque feu est modélisé par un système de transition à 2 états (rouge, vert). La composition parallèle des 2 systèmes de transition FC_1 et FC_2 détermine un système de transition noté $FC_1 ||| FC_2$, où l'opérateur $|||$ est l'opérateur d'entrelacement.

•

L'entrelacement repose sur le fait que l'effet de deux actions concurrentes indépendantes α et β est identique à l'effet produit lorsque α et β sont exécutées successivement et dans un ordre arbitraire. Cela peut être noté formellement :

$$Effet(\alpha ||| \beta, \eta) = Effet\big((\alpha; \beta) + (\beta; \alpha), \eta\big),$$

où le point-virgule « ; » représente l'exécution séquentielle et l'opérateur « + » un choix non déterministe.

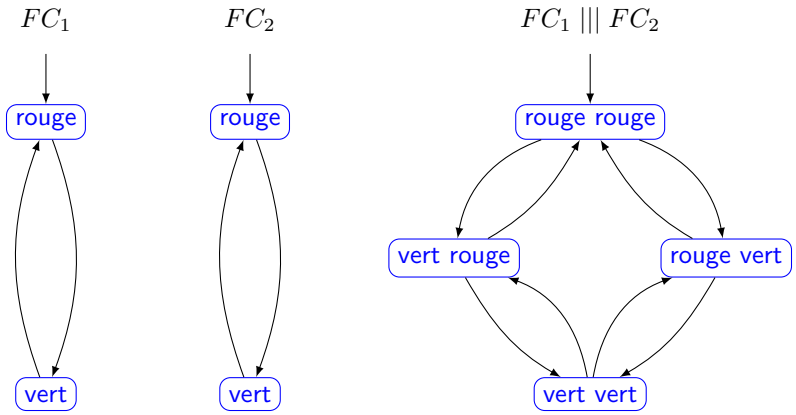

Figure 1.4. *Exemple d'entrelacement de 2 systèmes de transition.*

Exemple 1.5. On considère deux actions α et β qui modifient la valeur de deux variables x et y respectivement : l'action α est déterminée par $x \mapsto x+1$ et l'action β est déterminée par $y \mapsto y-2$. Le système de transition obtenu, lorsqu'initialement $x=0$ et $y=7$, est représenté sur la figure 1.5.

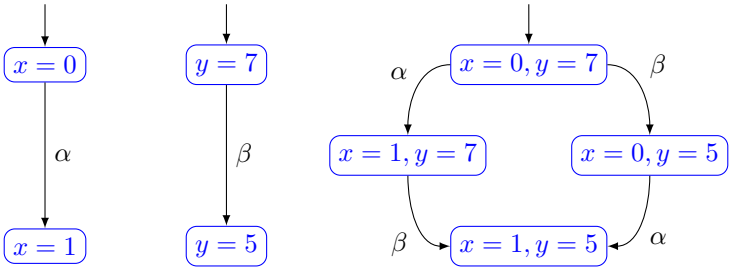

Figure 1.5. *Entrelacement de 2 actions indépendantes sur des variables distinctes.*

•

On peut donc définir l'entrelacement de 2 systèmes de transition. On suppose ici que les deux systèmes n'admettent pas de variable commune.

Définition 1.11 (Entrelacement de systèmes de transition)**.** Soient $ST_1 = (S_1, Act_1, \rightarrow_1, I_1, Prop_1, L_1)$ et $ST_2 = (S_2, Act_2, \rightarrow_2, I_2, Prop_2, L_2)$ deux systèmes de transition. Le système de transition $ST_1 ||| ST_2$ est défini par

$$ST_1 ||| ST_2 = (S_1 \times S_2, Act_1 \cup Act_2, \rightarrow, I_1 \times I_2, Prop_1 \cup Prop_2, L),$$

où la relation de transition $\rightarrow$ est définie par les règles

$$\frac{s_1 \xrightarrow{\alpha}_1 s_1'}{< s_1, s_2 > \xrightarrow{\alpha} < s_1', s_2 >} \quad \text{et} \quad \frac{s_2 \xrightarrow{\alpha}_2 s_2'}{< s_1, s_2 > \xrightarrow{\alpha} < s_1, s_2' >},$$

et où la fonction d'étiquetage L est définie par

$$L(< s_1, s_2 >) = L_1(s_1) \cup L_2(s_2).$$

□

1.5.2 Entrelacement de graphes conditionnels

Lorsque deux systèmes partagent des variables, l'entrelacement $|||$ de systèmes de transition peut conduire à des états paradoxaux.

Exemple 1.6. Considérons les actions α et β déterminées par $x \mapsto 2x$ et $x \mapsto x+1$ respectivement. L'entrelacement de ces 2 actions, à partir de l'état initial $< x = 3, x = 3 >$, produit l'état paradoxal $< x = 6, x = 4 >$, illustré sur la figure 1.6.

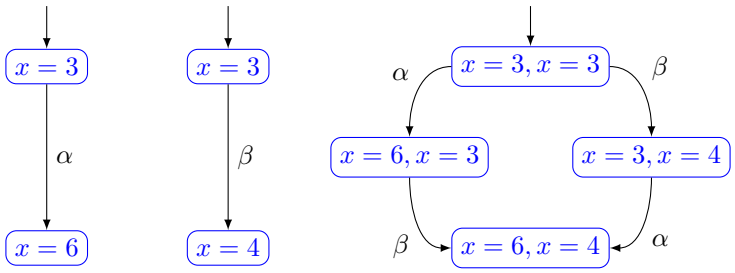

Figure 1.6. *État paradoxal produit par l'entrelacement de 2 systèmes qui admettent une variable en commun.*

•

On définit donc l'opérateur d'entrelacement $|||$ pour des graphes conditionnels.

Définition 1.12 (Entrelacement de graphes conditionnels)**.** Soient deux graphes conditionnels $GC_1 = (Loc_1, Act_1, \mathit{Effet}_1, \hookrightarrow_1, Loc_{0,1}, g_{0,1})$ et $GC_2 = (Loc_2, Act_2, \mathit{Effet}_2, \hookrightarrow_2, Loc_{0,2}, g_{0,2})$ sur des ensembles de variables Var_1 et Var_2 respectivement. Le graphe conditionnel $GC_1 \,|||\, GC_2$ est défini sur $Var_1 \cup Var_2$ par

$$GC_1 \,|||\, GC_2 = (Loc_1 \times Loc_2, Act_1 \uplus Act_2, \mathit{Effet}, \hookrightarrow, Loc_{0,1} \times Loc_{0,2}, g_{0,1} \wedge g_{0,2}),$$

où la relation de transition conditionnelle $\hookrightarrow$ est définie par les règles

$$\frac{l_1 \overset{g:\alpha}{\hookrightarrow}_1 l_1'}{< l_1, l_2 > \overset{g:<\alpha,1>}{\hookrightarrow} < l_1', l_2 >} \quad \text{et} \quad \frac{l_2 \overset{g:\alpha}{\hookrightarrow}_2 l_2'}{< l_1, l_2 > \overset{g:<\alpha,2>}{\hookrightarrow} < l_1, l_2' >},$$

et où la fonction d'effet Effet est définie par

$$\mathit{Effet}(\alpha, \eta)(v) = \begin{cases} \mathit{Effet}_i(< \alpha, i >, \eta_{|\, Var_i})(v), \text{ si } v \in Var_i, \\ \eta(v) \text{ sinon,} \end{cases}$$

où $\eta_{|\, Var_i}$ désigne la restriction de η à Var_i. □

Les graphes conditionnels GC_1 et GC_2 ont les variables appartenant à $Var_1 \cap Var_2$ en commun. Ces variables communes sont dites *globales*. Les variables appartenant à $Var_1 \setminus Var_2$ sont les variables locales de GC_1 et les variables appartenant à $Var_2 \setminus Var_1$ sont les variables locales de GC_2. L'opérateur $\uplus$ désigne l'union disjointe. Il n'y a donc pas d'action partagée dans le graphe conditionnel $GC_1 \,|||\, GC_2$ obtenu par entrelacement.

Remarque. Soient GC_1 et GC_2 deux graphes conditionnels. On note $ST(GC_1)$ et $ST(GC_2)$ les systèmes de transition engendrés par GC_1 et GC_2 respectivement. On ne doit pas confondre les systèmes de transition

$$ST(GC_1) \,|||\, ST(GC_2)$$

et

$$ST\big(GC_1 \,|||\, GC_2\big).$$

Le système de transition $ST(GC_1) \;|||\; ST(GC_2)$ est défini pour des graphes conditionnels qui ne partagent pas de variable, alors que le système de transition $ST\big(GC_1 \;|||\; GC_2\big)$ est défini même si les graphes conditionnels admettent des variables communes. ◁

Remarque (Sur l'atomicité). L'opérateur d'entrelacement pour des graphes conditionnels nécessite que les actions soient indivisibles. Par exemple, une action définie par

$$x \mapsto x+1, y \mapsto 2x+1 \text{ et } z \mapsto (x-z)^2 \times y \text{ si } x < 12,$$

doit être exécutée complètement. Les sous-actions ne doivent pas être dissociées dans le processus d'entrelacement. Une proposition est donc *atomique* lorsqu'elle peut être représentée par une seule étiquette le long d'une arête. ◁

1.5.3 Opérateur de composition

On a défini un opérateur d'entrelacement $|||$ pour des systèmes de transition sans variable partagée, et pour des graphes conditionnels qui peuvent admettre des variables en commun.

Nous allons maintenant définir un opérateur pour des processus concurrents qui interagissent par échange de message. Ces échanges impliquent que les sous-systèmes sont (au moins partiellement) synchronisés.

On considère alors un sous-ensemble d'actions partagées $H \subseteq Act_1 \cap Act_2$ et une action $\tau \notin H$. Les actions qui ne sont pas dans H (par exemple, l'action τ) sont indépendantes et peuvent être exécutées par le principe d'entrelacement.

Définition 1.13 (Opérateur d'échange synchrone)**.** Soient $ST_1 = (S_1, Act_1, \to_1, I_1, Prop_1, L_1)$ et $ST_2 = (S_2, Act_2, \to_2, I_2, Prop_2, L_2)$ deux systèmes de transition. Soit $H \subseteq Act_1 \cap Act_2$ et $\tau \notin H$. Le système de transition $ST_1 \parallel_H ST_2$ est défini par

$$ST_1 \parallel_H ST_2 = (S_1 \times S_2, Act_1 \cup Act_2, \to, I_1 \times I_2, Prop_1 \cup Prop_2, L),$$

où la fonction d'étiquetage L est définie par

$$L(< s_1, s_2 >) = L_1(s_1) \cup L_2(s_2),$$

et où la relation de transition $\to$ est définie par les deux règles suivantes :

- règle d'entrelacement pour $\alpha \notin H$:

$$\frac{s_1 \xrightarrow{\alpha}_1 s_1'}{< s_1, s_2 > \xrightarrow{\alpha} < s_1', s_2 >} \quad \text{et} \quad \frac{s_2 \xrightarrow{\alpha}_2 s_2'}{< s_1, s_2 > \xrightarrow{\alpha} < s_1, s_2' >},$$

- règle d'échange synchrone pour $\alpha \in H$:

$$\frac{s_1 \xrightarrow{\alpha}_1 s_1' \;\wedge\; s_2 \xrightarrow{\alpha}_2 s_2'}{< s_1, s_2 > \xrightarrow{\alpha} < s_1', s_2' >}.$$

□

Remarques.

- La condition $\ll \tau \notin H \gg$ signifie que H ne couvre pas totalement l'ensemble d'actions $Act_1 \cup Act_2$ (ce qui pourrait se produire dans le cas $Act_1 = Act_2$).
- Lorsque $H = (Act_1 \cap Act_2) \setminus \{\tau\}$, on écrit $\parallel$ à la place de $\parallel_H$.
- Lorsque $H = \varnothing$, on a

$$ST_1 \parallel_\varnothing ST_2 = ST_1 \;|||\; ST_2.$$

◁

Remarque. L'opérateur $\|_H$ est commutatif mais il n'est pas associatif lorsque l'ensemble H varie. Plus précisément, on a en général

$$(ST_1 \|_H ST_2) \|_{H'} ST_3 \neq ST_1 \|_H (ST_2 \|_{H'} ST_3)$$

si $H \neq H'$. Cependant, l'associativité est obtenue en fixant H :

$$(ST_1 \|_H ST_2) \|_H ST_3 = ST_1 \|_H (ST_2 \|_H ST_3),$$

ce qui permet de définir la composition d'un nombre fini de systèmes de transition

$$ST_1 \|_H ST_2 \|_H \cdots \|_H ST_n,$$

où $H \subseteq Act_1 \cap \cdots \cap Act_n$. ◁

Exemple 1.7 (Exclusion mutuelle par sémaphore)**.** Considérons deux processus P_1, P_2 admettant les états critiques respectifs crit_1, crit_2, et partageant le sémaphore binaire[1] y. Ces processus sont représentés par des graphes conditionnels sur la figure 1.7.

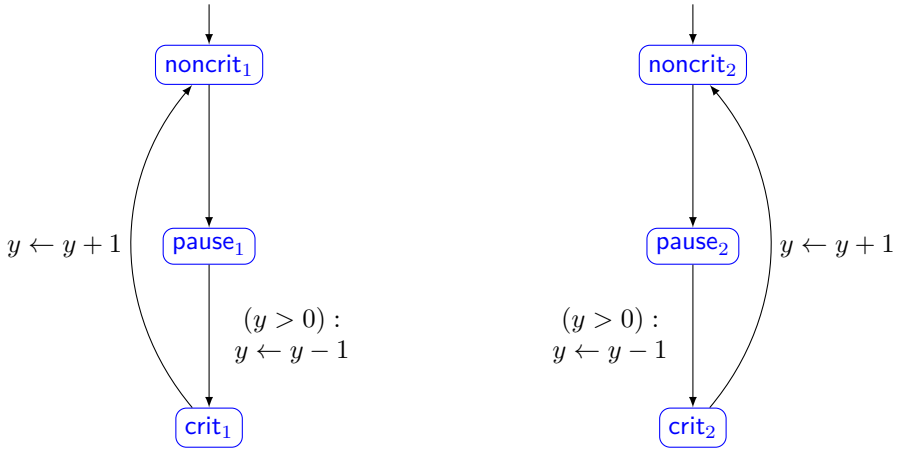

Figure 1.7. *Processus concurrents partageant le sémaphore binaire y.*

Le graphe conditionnel $P_1 \,|||\, P_2$ admet alors 9 lieux, dont le lieu indésirable $< \mathsf{crit}_1, \mathsf{crit}_2 >$. Le système de transition ST_{sem} obtenu par déploiement de $P_1 \,|||\, P_2$ admet 18 états, dont 8 seulement sont accessibles. En particulier, les états $< \mathsf{crit}_1, \mathsf{crit}_2, y = 0 >$ et $< \mathsf{crit}_1, \mathsf{crit}_2, y = 1 >$ sont inaccessibles, ce qui signifie que les processus P_1 et P_2 ne peuvent pas être en état critique simultanément. Le système de transition ST_{sem} satisfait donc la propriété d'*exclusion mutuelle.* •

1.6 Explosion de l'espace de phase

Pour de nombreux systèmes, le nombre d'états est fini, mais très grand, ce qui constitue un obstacle pour l'exécution des algorithmes de vérification qui explorent les états du système.

Il existe heureusement des méthodes pour réduire la taille de l'espace de phase, ou pour optimiser l'exécution des algorithmes.

Ainsi, considérons un graphe conditionnel $(Loc, Act, Effet, \hookrightarrow, Loc_0, g_0)$ sur un ensemble de variables Var. Le nombre d'états du système de transition correspondant est alors

$$|Loc| \times \prod_{x \in Var} |dom(x)| \,.$$

1. Un sémaphore est une variable partagée par différents processus, qui garantit que ceux-ci ne peuvent y accéder que de façon séquentielle à travers des opérations atomiques.

Le nombre d'états croît donc exponentiellement avec le nombre de variables. Pour N variables admettant chacune k valeurs possibles, ce nombre est de l'ordre de k^N.

Par exemple, si un graphe conditionnel admet 10 lieux, 3 variables booléennes et 5 variables entières dont les valeurs sont comprises entre 0 et 9, alors le système admet $10 \times 2^3 \times 10^5$ états, soit 8000000 états !

1.7 Exercices

Exercice 1. Le distributeur de boissons

On considère un distributeur de boissons, qui délivre du café et des sodas. Ce distributeur de boissons détermine un système de transition DB qui est illustré sur la figure 1.8 ci-dessous.

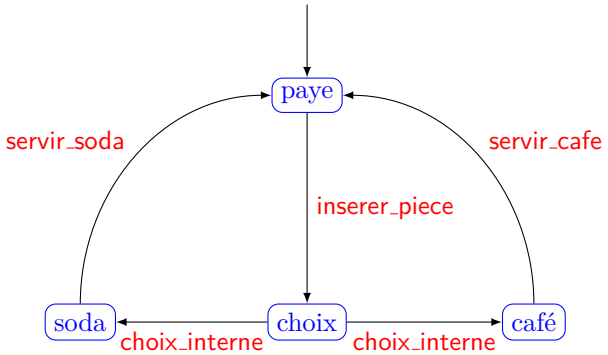

Figure 1.8. *Système de transition modélisant un distributeur de boissons.*

1. On considère l'ensemble de propositions atomiques $Prop = \{\mathsf{payé}, \mathsf{délivré}\}$. Détailler les éléments du système de transition DB.
2. Le système de transition DB est-il déterministe ?
3. On considère les fragments d'exécution suivants :

$$\rho_1 = \text{paye} \xrightarrow{\text{inserer\_piece}} \text{choix} \xrightarrow{\text{choix\_interne}} \text{café} \dots ,$$
$$\rho_2 = \text{café} \xrightarrow{\text{servir\_cafe}} \text{paye} \xrightarrow{\text{inserer\_piece}} \text{choix} \xrightarrow{\text{choix\_interne}} \text{café} \dots ,$$
$$\rho_3 = \text{paye} \xrightarrow{\text{inserer\_piece}} \text{choix} \xrightarrow{\text{choix\_interne}} \text{café} \xrightarrow{\text{servir\_cafe}} \text{paye} .$$

   Décrire ces trois fragments d'exécution. Ces fragments d'exécution sont-ils des exécutions ?
4. On considère ensuite une extension du système de transition DB, qui compte le nombre de sodas et le nombre de cafés, et qui renvoie les pièces insérées si la machine est vide. On souhaite modéliser ce distributeur par un graphe conditionnel GC. On pose alors $Loc = \{\mathsf{démarre}, \mathsf{choix}\}$, $Loc_0 = \{\mathsf{démarre}\}$ et

$$Act = \{\mathsf{inserer\_piece}, \mathsf{retourne\_piece}, \mathsf{recharge}, \mathsf{servir\_soda}, \mathsf{servir\_cafe}\}.$$

   On considère de plus l'ensemble de variables $Var = \{nsoda, ncafe\}$, où le domaine de chaque variable est $\{0, 1, \dots, max\}$.
   (a) Représenter les transitions du graphe conditionnel GC.
   (b) Déterminer l'effet de chaque action sur une évaluation η des variables $nsoda$ et $ncafe$.
   (c) Représenter le système de transition obtenu par déploiement du graphe conditionnel GC dans le cas $max = 2$.

Exercice 2. Circuit séquentiel

On considère un circuit séquentiel, illustré sur la figure 1.9 ci-dessous, avec une variable d'entrée x, une variable de sortie y et un enregistrement r. La fonction de contrôle pour la variable y est donnée par $\lambda_y = \neg(x \oplus r)$, où $\oplus$ désigne le « ou » exclusif (XOR). L'évaluation de l'enregistrement est modifiée par la fonction de circuit définie par $\delta_r = x \vee r$.

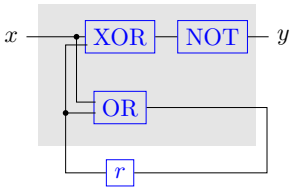

Figure 1.9. *Circuit séquentiel* hardware.

1. Représenter ce circuit comme un système de transition.
2. Détailler les éléments constitutifs de ce système de transition.

Exercice 3. Feux de circulation

1. On considère deux feux de circulation placés sur des routes qui ne se coupent pas. Chaque feu est modélisé par un système de transition à 2 états (rouge, vert), comme illustré sur la figure 1.10 ci-dessous.

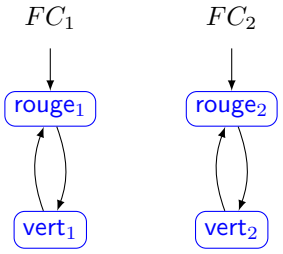

Figure 1.10. *Feux de circulation indépendants.*

   Décrire la composition par entrelacement $FC_1 \,|||\, FC_2$ des 2 systèmes de transition FC_1 et FC_2.
2. On considère maintenant deux feux de circulation correctement synchronisés.

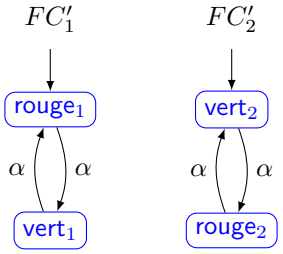

Figure 1.11. *Feux de circulation totalement synchronisés.*

   Décrire la composition parallèle $FC'_1 \parallel FC'_2$ des 2 systèmes de transition FC'_1 et FC'_2.

Exercice 4. Entrelacement de graphes conditionnels

On considère deux graphes conditionnels GC_1 et GC_2 partageant une même variable x, représentés sur la figure 1.12 ci-dessous.

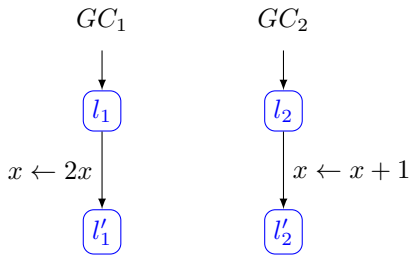

Figure 1.12. *Graphes conditionnels GC_1 et GC_2 partageant une même variable x.*

1. Construire le graphe conditionnel d'entrelacement $GC_1 \,|||\, GC_2$.
2. Construire le système de transition obtenu par déploiement du graphe conditionnel $GC_1 \,|||\, GC_2$, avec comme condition initiale $x = 3$.

Exercice 5. Exclusion mutuelle par sémaphore

On considère deux processus P_1, P_2 admettant les états critiques respectifs crit_1, crit_2, et partageant le sémaphore binaire y. Ces processus sont représentés par des graphes conditionnels sur la figure 1.13.

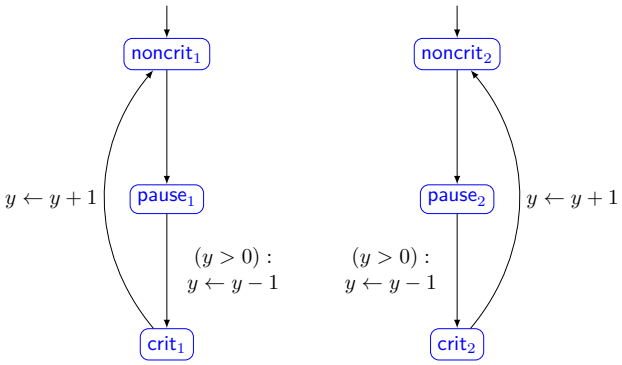

Figure 1.13. *Processus concurrents partageant le sémaphore binaire y.*

1. Construire le graphe conditionnel d'entrelacement $P_1 \,|||\, P_2$.
2. Construire le système de transition ST_{sem} obtenu par déploiement de $P_1 \,|||\, P_2$.

CHAPITRE 2

Propriétés linéaires

Lorsqu'on étudie le comportement d'un système réel, après que l'étape de modélisation de ce système a été réalisée, le processus de vérification de certaines des propriétés de son modèle peut être entamé. Ce processus de vérification nécessite alors une spécification des propriétés intéressantes (c'est-à-dire une description précise et non ambiguë de ce que ce système doit faire). Nous allons découvrir dans le deuxième chapitre de ce cours une classe importante de propriétés : les propriétés *linéaires*.

2.1 Comportement linéaire

On peut analyser le comportement d'un système de transition en observant ses états ou en observant ses actions (ou encore en combinant les deux approches). Dans cette partie, on choisit la première approche, pour laquelle on privilégie l'observation des états et des propositions atomiques qui leur sont associées.

2.1.1 Chemins et graphes d'états

Soit $ST = (S, Act, \rightarrow, I, Prop, L)$ un système de transition.

Définition 2.1 (Graphe d'états)**.** Le graphe d'états de ST, noté $G(ST)$, est le graphe orienté (V, E) avec pour ensemble de sommets $V = S$ et pour ensemble d'arêtes

$$E = \{(s, s') \in S \times S \mid s' \in Post(s)\}.$$

□

Le graphe d'états $G(ST)$ est donc obtenu en omettant les propositions atomiques associées aux états de ST, et en ignorant quels sont les états initiaux possibles. D'éventuelles transitions multiples entre 2 états du système de transition (associées à des actions différentes) sont représentées dans le graphe d'états par une seule arête.

Pour un graphe d'états, on définit les ensembles de successeurs et de prédécesseurs : $Post^*(s)$, $Post^*(C)$, $Pre^*(s)$, $Pre^*(C)$ pour $s \in S$ et $C \subseteq S$. Ainsi, $Post^*(s)$ est l'ensemble des états accessibles dans $G(ST)$ depuis un état s. L'étoile permet de différencier les ensembles définis à partir du système de transition des ensembles définis à partir de son graphe d'états. L'ensemble $Access(ST)$ des états accessibles est égal à $Post^*(I)$.

Remarque. Dans les définitions des ensembles $Post(s)$, $Post(C)$, $Pre(s)$, $Pre(C)$ du premier chapitre, on a considéré les successeurs et les prédécesseurs *directs*. Pour les ensembles $Post^*(s)$, $Post^*(C)$, $Pre^*(s)$, $Pre^*(C)$, on considère cette fois tous les successeurs et tous les prédécesseurs. ◁

On décrit le comportement d'un système de transition par l'étude des chemins dans son graphe d'états.

Définition 2.2 (Fragment de chemin)**.** Un fragment de chemin fini $\hat{\pi}$ de ST est une suite finie d'états

$$\hat{\pi} = s_0 s_1 \dots s_n,$$

telle que $s_i \in Post(s_{i-1})$ pour tout $i \in \{1, \dots, n\}$, avec $n \in \mathbb{N}$.

Un fragment de chemin infini π de ST est une suite infinie d'états

$$\pi = s_0 s_1 s_2 \dots,$$

telle que $s_i \in Post(s_{i-1})$ pour tout $i \geq 1$. □

Pour $j \geq 0$, on note $\hat{\pi}[j] = s_j$ et $\pi[j] = s_j$. Le premier élément $\hat{\pi}[0] = s_0$ (ou $\pi[0] = s_0$) est parfois noté $first(\hat{\pi})$ (ou $first(\pi)$). Les préfixes sont notés $\hat{\pi}[..j] = s_0 \dots s_j$ (ou $\pi[..j] = s_0 \dots s_j$) ; les suffixes sont notés $\hat{\pi}[j..]$ (ou $\pi[j..]$). Pour un fragment de chemin fini $\hat{\pi}$, on note le dernier élément $last(\hat{\pi}) = s_n$ et la longueur $n = len(\hat{\pi})$. Pour un fragment de chemin infini, on note $last(\pi) = \perp$ (ce qui signifie « indéfini ») et $len(\pi) = \infty$.

Définition 2.3 (Fragment de chemin maximal, fragment de chemin initial, chemin)**.** Un fragment de chemin *maximal* est soit un fragment de chemin fini qui termine dans un état final, soit un fragment de chemin infini.

Un fragment de chemin est dit *initial* s'il commence dans un état initial.

Un *chemin* est un fragment de chemin initial et maximal. □

Remarque. On ne doit pas confondre la notion de chemin dans un graphe avec la notion de chemin dans un système de transition. Dans un système de transition, un chemin est maximal, pas nécessairement pas dans un graphe. De plus, dans un système de transition, un chemin peut être infini, mais pas dans un graphe. ◁

Si s est un état d'un système de transition ST, on note $Paths(s)$ l'ensemble des fragments de chemins maximaux qui commencent en s et $Paths_{fin}(s)$ l'ensemble des fragments de chemins finis qui commencent en s. Enfin, on note $Paths(ST)$ l'ensemble des chemins dans ST et $Paths_{fin}(ST)$ l'ensemble des fragments de chemins finis initiaux dans ST.

2.1.2 Rappels sur les graphes

Dans cette section, on propose quelques rappels sur la théorie des graphes.

Définition 2.4 (Graphe)**.** Un *graphe orienté* (ou simplement un *graphe*) est une paire $G = (V, E)$ formée par un ensemble V de *sommets* et une relation $E \subseteq V \times V$ définissant un ensemble d'arêtes. Les éléments de E sont appelés *arcs* (pour un graphe orienté) ou *arêtes* (par abus de langage). □

Matrice d'adjacence

Soit $G = (V, E)$ un graphe, dont l'ensemble V est ordonné sous la forme d'une suite $(s_1, s_2, \dots, s_N)$. La matrice d'adjacence M du graphe G est la matrice carrée de taille N définie par :

$$M_{i,j} = \begin{cases} 1 & \text{si } (s_i, s_j) \text{ est un arc,} \\ 0 & \text{sinon.} \end{cases}$$

Listes d'adjacence

On peut également représenter un graphe par listes d'adjacence, en associant à chaque sommet une suite ordonnée de tous ses successeurs.

Algorithmes de balayage

L'exploration d'un graphe est souvent effectuée par l'exécution d'un algorithme de balayage de type *Depth-First Search* (DFS) ou *Breadth-First Search* (BFS), qui reposent eux-mêmes sur l'algorithme 1, permettant de déterminer les sommets accessibles depuis un sommet s_0. Dans l'algorithme 1, l'ensemble R donne les sommets qui ont été visités, et l'ensemble U garde une trace des sommets qui sont à explorer (pourvu qu'ils ne soient pas dans R). Chaque sommet s' peut être ajouté à U au plus $|Pre(s')|$ fois. Donc l'algorithme 1 termine après au plus $\mathcal{O}\big(M\big)$ itérations, où $M = |E|$ est le nombre d'arêtes. Lorsque tous les sommets doivent être visités, la complexité temporelle est de l'ordre de $\mathcal{O}(N + M)$, où $N = |V|$.

Algorithme 1 Analyse d'accessibilité

Entrée : graphe $G = (V, E)$, sommet $s_0 \in V$.
Sortie : ensemble $Access(s_0)$ des sommets accessibles depuis s_0.

```
Ensemble de sommets R ← ∅                          // l'ensemble des états accessibles
Ensemble de sommets U ← {s_0}                      // sommets à explorer
// initialement, aucune arête n'est marquée
Tant que U ≠ ∅ faire
    choisir s ∈ U                                  // on choisit arbitrairement un état s dans U
    Si ∃s' ∈ Post(s) tel que (s, s') n'est pas marquée alors
        choisir un tel sommet s' ∈ Post(s)
        marquer l'arête (s, s')
        Si s' ∉ R alors
            U ← U ∪ {s'}                           // on s'assure que tous les successeurs
            R ← R ∪ {s'}                           // de s seront visités
        Fin Si
    Sinon
        U ← U \ {s}
    Fin Si
Fin Tant que
renvoyer R                                         // R = Access(s_0)
```

L'algorithme DFS effectue une exploration en « profondeur », en structurant U comme une pile (qui satisfait la règle « *Last In First Out* »). La pile U admet les opérations $top(U)$, qui renvoie le premier élément de U (celui qui se trouve au sommet de la pile), $pop(U)$, qui efface le premier élément de U, et $push(s, U)$, qui dépose l'élément s au sommet de la pile U.

Au contraire, l'algorithme BFS effectue une exploration en « largeur », en structurant U comme une file (qui satisfait la règle « *First In First Out* »).

Exemple 2.1. On considère le graphe représenté sur la figure 2.1.

Les listes d'adjacence de ce graphe sont données dans le tableau 2.1.

Lorsqu'on exécute l'algorithme DFS sur le graphe de la figure 2.1, à partir du sommet 1, l'ensemble R et la pile U prennent successivement les valeurs données dans le tableau 2.2. •

Rappel. Pour décrire la complexité temporelle d'un algorithme, on utilise les équivalents asymptotiques $\mathcal{O}$, Ω et Θ. Si f et g sont deux fonctions définies sur $\mathbb{N}$, alors $g = \mathcal{O}(f)$ signifie qu'il existe une

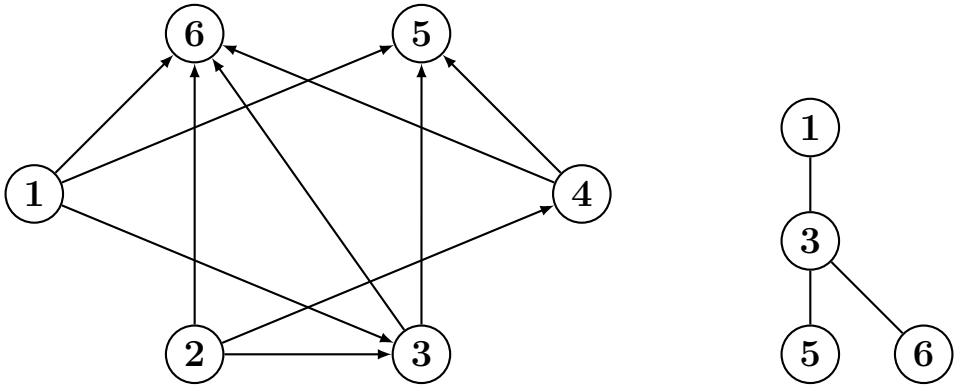

Figure 2.1. *Exemple de graphe et de l'arborescence obtenue en appliquant l'algorithme DFS en partant du sommet 1.*

Tableau 2.1. *Listes d'adjacence du graphe représenté sur la figure 2.1.*

Sommet	Successeurs
1	$\{3,5,6\}$
2	$\{3,4,6\}$
3	$\{5,6\}$
4	$\{5,6\}$
5	$\varnothing$
6	$\varnothing$

Tableau 2.2. *Valeurs successives de l'ensemble R et de la pile U pour un balayage du graphe de la figure 2.1 par l'algorithme DFS à partir du sommet 1.*

$R=\varnothing$	$R=\{3\}$	$R=\{3,5\}$	$R=\{3,5,6\}$	$R=\{3,5,6\}$	$R=\{3,5,6\}$	$R=\{3,5,6\}$
1	3 1	5 3 1	6 3 1	3 1	1	$\varnothing$
U	U	U	U	U	U	U

constante $C>0$ et un entier $N>0$ tels que $g(n)\leq C\times f(n)$ pour $n\geq N$. Inversement, $g=\Omega(f)$ signifie qu'il existe une constante $C>0$ et un entier $N>0$ tels que $g(n)\geq C\times f(n)$ pour $n\geq N$. Enfin, $g=\Theta(f)$ signifie que l'on a à la fois $g=\mathcal{O}(f)$ et $g=\Omega(f)$. ◦

2.1.3 Traces

Dans le premier chapitre, nous avons défini les exécutions d'un système de transition, qui sont des suites avec une alternance d'états et d'actions. Dans la suite du cours, on s'intéresse surtout aux états d'un système de transition. Comme ces états sont parfois difficilement observables, on s'intéresse plutôt à leurs propositions atomiques. On considère ainsi des suites du type

$$L(s_0)L(s_1)L(s_2)\dots.$$

De telles suites sont appelées *traces*. Les traces d'un système de transition sont alors vues comme des mots sur l'alphabet 2^{Prop}. Dans toute la suite, on supposera que le système de transition n'a pas d'état final. Dans ce cas, les traces pourront être des mots infinis.

Remarque. Un système de transition qui admet des états finaux peut toujours être modifié pour ne plus en admettre ; il suffit pour cela d'ajouter des états à la suite des états finaux et de les équiper d'une boucle. ◁

Définition 2.5 (Trace)**.** Soit $ST = (S, Act, \to, I, Prop, L)$ un système de transition sans état final. La trace d'un fragment de chemin infini $\pi = s_0 s_1 s_2 \dots$ est définie par

$$trace(\pi) = L(s_0)L(s_1)L(s_2)\dots.$$

La trace d'un fragment de chemin fini $\hat{\pi} = s_0 s_1 \dots s_n$ est définie par

$$trace(\hat{\pi}) = L(s_0)L(s_1)\dots L(s_n).$$

□

La trace d'un fragment de chemin est donc un mot fini ou infini sur l'alphabet $\Sigma = 2^{Prop}$. Si Π est un ensemble de chemins, on pose

$$trace(\Pi) = \{trace(\pi) \mid \pi \in \Pi\}.$$

Une trace (infinie) d'un état $s \in S$ est la trace d'un fragment de chemin infini π tel que $first(\pi) = s$. De la même façon, une trace finie d'un état s est la trace d'un fragment de chemin fini qui commence en s. On note alors $Traces(s)$ l'ensemble des traces d'un état s et finalement :

$$Traces(ST) = \bigcup_{s \in I} Traces(s).$$

Remarques. Si *Prop* admet n éléments ($n \geq 1$), alors l'alphabet $\Sigma = 2^{Prop}$ en admet 2^n. Par exemple, si $Prop = \{a, b, c\}$, alors

$$\Sigma = \big\{\varnothing, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}\big\}.$$

Si s est un état, alors $L(s)$ est un sous-ensemble de *Prop*, qui peut posséder plusieurs éléments. Ainsi, les « lettres » $L(s_0)$, $L(s_1)$, $L(s_2)$, … d'une trace sont de « grosses lettres ». On peut avoir par exemple :

$$L(s_0) = \{x, y, z\}, \quad L(s_1) = \{w, z\}, \quad L(s_2) = \{t, x, y\},$$

ce qui donnerait

$$L(s_0)L(s_1)L(s_2)\cdots = \{x, y, z\}\,\{w, z\}\,\{t, x, y\}\dots.$$

◁

On définit de la même façon les traces finies d'un système de transition, à partir de ses chemins finis :

$$Traces_{fin}(ST) = \bigcup_{s \in I} Traces_{fin}(s).$$

2.1.4 Rappels sur les langages

L'étude des traces d'un système de transition suppose une bonne connaissance des propriétés élémentaires des langages et de leurs opérations. Nous rappelons ici quelques-unes de ces propriétés.

Un *alphabet* est un ensemble fini non vide Σ. Les éléments de Σ sont appelés *symboles* ou *lettres*. Un *mot* sur Σ est une suite finie ou infinie de symboles de Σ. Un mot fini est donc de la forme

$$w = A_1 A_2 \dots A_n,$$

avec $n \in \mathbb{N}$ et $A_i \in \Sigma$ pour tout $i \in \{1, \dots, n\}$, et un mot infini est de la forme

$$\sigma = A_1 A_2 A_3 \dots.$$

Le mot obtenu avec $n = 0$ est appelé *mot vide* et est souvent noté ε. Un mot fini $w = A_1A_2 \dots A_n$ a pour longueur n. Un mot infini a une longueur infinie souvent notée ω. On note Σ^* l'ensemble des mots finis sur Σ, et Σ^ω l'ensemble des mots infinis sur Σ. On a $\varepsilon \in \Sigma^*$. On note $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$ l'ensemble des mots finis non vides. On appelle *langage* tout ensemble $\mathcal{L}$ de mots finis sur l'alphabet Σ. On a donc $\mathcal{L} \subseteq \Sigma^*$.

Remarque. Ne pas confondre le mot vide ε et le langage vide $\mathcal{L} = \varnothing$. ◁

Un *préfixe* d'un mot fini $w = A_1A_2 \dots A_n$ est un mot de la forme $v = A_1A_2 \dots A_i$ avec $0 \leq i \leq n$ ($v = \varepsilon$ si $i = 0$). Un *suffixe* d'un mot fini $w = A_1A_2 \dots A_n$ est un mot de la forme $v = A_iA_{i+1} \dots A_n$ avec $1 \leq i \leq n+1$ ($v = \varepsilon$ si $i = n+1$). Le mot vide ε est donc préfixe et suffixe de tous les mots finis. Un mot de la forme $A_i \dots A_j$ avec $1 \leq i \leq j \leq n$ est appelé *sous-mot* de w. On définit de même les préfixes, suffixes et sous-mots d'un mot infini (un suffixe d'un mot infini est lui-même infini).

Opérations sur les mots

La *concaténation* et la *répétition finie* sont deux opérations importantes sur les mots.

La concaténation de 2 mots agit en « collant » deux mots l'un à la suite de l'autre. Par exemple, la concaténation des mots BA et AAB est le mot $BA.AAB = BAAAB$. La concaténation d'un mot w avec lui-même est notée w^2. On généralise cette opération pour définir w^n avec n entier naturel. On a en particulier $w^0 = \varepsilon$ et $w^1 = w$.

La répétition finie (également appelée *Kleene star* en anglais), notée avec une étoile *, d'un mot fini w, produit le langage

$$w^* = \{w^i \mid i \in \mathbb{N}\}.$$

Par exemple, on a

$$(AB)^* = \{\varepsilon, AB, ABAB, ABABAB, \dots\}.$$

Le mot vide ε appartient à w^* pour chaque mot w. On note également

$$w^+ = \{w^i \mid i \geq 1\},$$

où de façon équivalente : $w^+ = w^* \setminus \{\varepsilon\}$.

Opérations sur les langages

On définit également la concaténation pour les langages. Pour $\mathcal{L}_1, \mathcal{L}_2 \subseteq \Sigma^*$, on pose ainsi

$$\mathcal{L}_1.\mathcal{L}_2 = \{w_1.w_2 \mid w_1 \in \mathcal{L}_1, w_2 \in \mathcal{L}_2\}.$$

Par exemple, si $\mathcal{L}_1 = \{A, AB\}$ et $\mathcal{L}_2 = \{\varepsilon, BBB\}$, alors on a :

$$\mathcal{L}_1.\mathcal{L}_2 = \{A, AB, ABBB, ABBBB\},$$
$$\mathcal{L}_1^2 = \{AA, AAB, ABAB, ABA\}.$$

Pour $\mathcal{L}_1, \mathcal{L}_2 \subseteq \Sigma^*$, la réunion de $\mathcal{L}_1$ et $\mathcal{L}_2$ est définie par

$$\mathcal{L}_1 \cup \mathcal{L}_2 = \{w \in \Sigma^* \ ; \ w \in \mathcal{L}_1 \text{ ou } w \in \mathcal{L}_2\}.$$

On a alors

$$\mathcal{L} \cup \varnothing = \mathcal{L}, \quad \mathcal{L} \cup \Sigma^* = \Sigma^*, \quad \mathcal{L}_1 \cup \mathcal{L}_2 = \mathcal{L}_2 \text{ si } \mathcal{L}_1 \subseteq \mathcal{L}_2,$$

pour tous langages $\mathcal{L}$, $\mathcal{L}_1$, $\mathcal{L}_2$ sur Σ. On peut également définir l'intersection de deux langages, leur différence, le complémentaire d'un langage.

La concaténation de deux langages n'est pas commutative en général. On a néanmoins les propriétés suivantes :

$$\begin{aligned}
&\mathcal{L}(\mathcal{L}'\mathcal{L}'') = (\mathcal{L}\mathcal{L}')\mathcal{L}'',\\
&\mathcal{L}(\mathcal{L}' \cup \mathcal{L}'') = \mathcal{L}\mathcal{L}' \cup \mathcal{L}\mathcal{L}'',\\
&(\mathcal{L} \cup \mathcal{L}')\mathcal{L}'' = \mathcal{L}\mathcal{L}'' \cup \mathcal{L}'\mathcal{L}'',\\
&\mathcal{L}\{\varepsilon\} = \{\varepsilon\}\mathcal{L} = \mathcal{L},\\
&\varnothing\mathcal{L} = \mathcal{L}\varnothing = \varnothing,
\end{aligned}$$

pour tous langages $\mathcal{L}$, $\mathcal{L}'$, $\mathcal{L}''$ sur Σ.

On définit enfin les puissances et la répétition finie (étoile de Kleene) d'un langage $\mathcal{L}$:

$$\mathcal{L}^* = \bigcup_{i\geq 0} \mathcal{L}^i, \quad \mathcal{L}^+ = \bigcup_{i\geq 1} \mathcal{L}^i,$$

où $\mathcal{L}^i$ désigne la concaténation de $\mathcal{L}$ répétée i fois.

2.1.5 Propriétés linéaires

Les propriétés linéaires sont une classe de propriétés qui déterminent les traces qu'un système de transition doit admettre.

Définition 2.6 (Propriété linéaire)**.** On appelle *propriété linéaire* sur un ensemble de propositions atomiques *Prop* tout sous-ensemble (non vide) de $(2^{Prop})^\omega$. □

Dans cette définition, $(2^{Prop})^\omega$ désigne l'ensemble des mots obtenus par concaténation infinie de symboles dans 2^{Prop}. Une propriété linéaire est donc un langage de mots infinis sur l'alphabet 2^{Prop}. On considère uniquement des mots *infinis*, car on étudie des systèmes de transition *sans état final*. On définit ensuite la relation de satisfaction d'une propriété linéaire par un système de transition.

Définition 2.7 (Relation de satisfaction pour une propriété linéaire)**.** Soit P une propriété linéaire sur un ensemble de propositions atomiques *Prop* et soit $ST = (S, Act, \to, I, Prop, L)$ un système de transition.

On dit que ST *satisfait* P, et l'on note $ST \models P$, si $Traces(ST) \subseteq P$.

On dit qu'un état $s \in S$ satisfait P, et l'on note $s \models P$, si $Traces(s) \subseteq P$. □

Exemple 2.2. Considérons deux feux de circulation correctement synchronisés. Le système de transition correspondant est illustré sur la figure 2.2. On considère de plus l'ensemble de propositions atomiques

$$Prop = \{\mathsf{rouge}_1, \mathsf{vert}_1, \mathsf{rouge}_2, \mathsf{vert}_2\}.$$

On étudie alors les deux propriétés linéaires P_1 et P_2 définies par :

- P_1 : « *le premier feu est vert une infinité de fois* »,
- P_2 : « *les deux feux ne sont jamais verts en même temps* ».

La proposition P_1 contient par exemple les mots

$$\begin{aligned}
&\{\mathsf{rouge}_1, \mathsf{vert}_2\}\{\mathsf{vert}_1, \mathsf{rouge}_2\}\{\mathsf{rouge}_1, \mathsf{vert}_2\}\{\mathsf{vert}_1, \mathsf{rouge}_2\}\ldots\\
&\varnothing\{\mathsf{vert}_1\}\varnothing\{\mathsf{vert}_1\}\varnothing\{\mathsf{vert}_1\}\ldots\\
&\{\mathsf{rouge}_1, \mathsf{vert}_1\}\{\mathsf{rouge}_1, \mathsf{vert}_1\}\{\mathsf{rouge}_1, \mathsf{vert}_1\}\{\mathsf{rouge}_1, \mathsf{vert}_1\}\ldots\\
&\{\mathsf{vert}_1, \mathsf{vert}_2\}\{\mathsf{vert}_1, \mathsf{vert}_2\}\{\mathsf{vert}_1, \mathsf{vert}_2\}\{\mathsf{vert}_1, \mathsf{vert}_2\}\ldots.
\end{aligned}$$

En revanche, le mot

$$\{\mathsf{rouge}_1, \mathsf{vert}_1\}\{\mathsf{rouge}_1, \mathsf{vert}_1\}\varnothing\varnothing\varnothing\varnothing\ldots$$

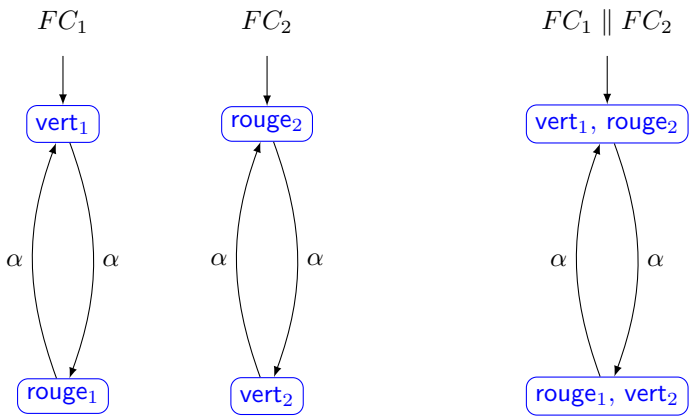

Figure 2.2. *Feux de circulation totalement synchronisés.*

n'est pas dans P_1 car il ne contient pas une infinité de fois vert_1.

Puis, la propriété P_2 contient par exemple les mots

$$\{\mathsf{rouge}_1, \mathsf{vert}_2\}\{\mathsf{vert}_1, \mathsf{rouge}_2\}\{\mathsf{rouge}_1, \mathsf{vert}_2\}\{\mathsf{vert}_1, \mathsf{rouge}_2\}\ldots$$
$$\varnothing\{\mathsf{vert}_1\}\varnothing\{\mathsf{vert}_1\}\varnothing\{\mathsf{vert}_1\}\ldots$$
$$\{\mathsf{rouge}_1, \mathsf{vert}_1\}\{\mathsf{rouge}_1, \mathsf{vert}_1\}\{\mathsf{rouge}_1, \mathsf{vert}_1\}\{\mathsf{rouge}_1, \mathsf{vert}_1\}\ldots,$$

mais le mot

$$\{\mathsf{rouge}_1, \mathsf{vert}_1\}\{\mathsf{rouge}_1, \mathsf{vert}_1\}\varnothing\varnothing\varnothing\varnothing\ldots$$

n'est pas dans P_2.

Il est clair que le système de transition modélisant les deux feux totalement synchronisés satisfait les propriétés P_1 et P_2. •

Remarque. Les propriétés linéaires (*linear-time properties* en anglais) intègrent de façon abstraite le déroulement temporel du comportement d'un système réel, ce qui permet notamment de tenir compte de l'ordre d'apparition des événements. Par exemple, une propriété comme « la voiture s'arrête une fois que le conducteur a freiné » nécessite de considérer une chronologie des événements. En revanche, l'aspect quantitatif du déroulement temporel est ignoré ici. Il peut être néanmoins pris en compte, notamment en considérant des systèmes de transition *temporisés*, qui sont des systèmes de transition équipés d'horloges. ◁

2.1.6 Traces équivalentes

Les propriétés linéaires décrivent les traces infinies qu'un système de transition doit admettre. Si deux systèmes de transition ST et ST' admettent les mêmes traces, on s'attend à ce qu'elles vérifient les mêmes propriétés. Le théorème suivant établit une caractérisation de l'inclusion de traces.

Théorème 2.1 (Inclusion de traces)**.** Soient ST et ST' deux systèmes de transition sans état final et admettant le même ensemble de propositions atomiques. Alors les deux assertions suivantes sont équivalentes :

- $\mathit{Traces}(ST) \subseteq \mathit{Traces}(ST')$;
- pour toute propriété linéaire P, $ST' \models P$ implique $ST \models P$.

Démonstration. (1) $\Rightarrow$ (2). Soit P une propriété linéaire telle que $ST' \models P$. Alors par définition, on a $\mathit{Traces}(ST') \subseteq P$. Puisque $\mathit{Traces}(ST) \subseteq \mathit{Traces}(ST')$, on en déduit que $\mathit{Traces}(ST) \subseteq P$. Donc $ST \models P$.

(2) $\Rightarrow$ (1). Considérons la propriété $P = \mathit{Traces}(ST')$. On a $\mathit{Traces}(ST') \subseteq \mathit{Traces}(ST') = P$, c'est-à-dire $ST' \models P$; donc $ST \models P$ d'après l'hypothèse (2). On obtient par définition $\mathit{Traces}(ST) \subseteq P$, ce qui est équivalent à $\mathit{Traces}(ST) \subseteq \mathit{Traces}(ST')$. □

On obtient le corollaire suivant.

Corollaire 2.1 (Traces équivalentes)**.** Soient ST et ST' deux systèmes de transition sans état final et admettant le même ensemble de propositions atomiques. Alors $\mathit{Traces}(ST) = \mathit{Traces}(ST')$ si et seulement si ST et ST' vérifient les mêmes propriétés linéaires.

Exemple 2.3. Considérons deux systèmes de transition obtenus en modélisant des distributeurs de boissons, représentés sur la figure 2.3.

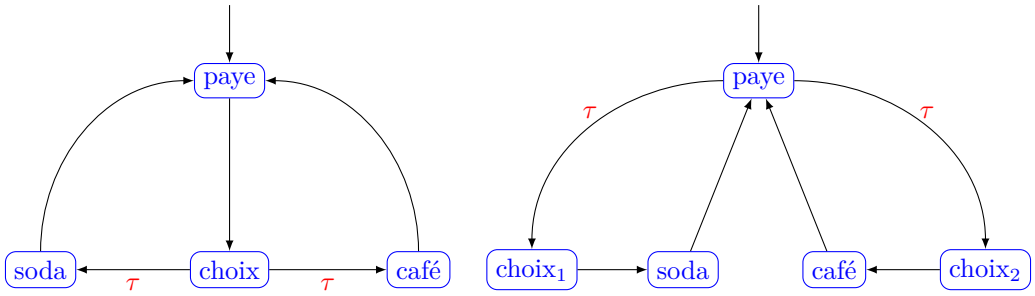

Figure 2.3. *Deux systèmes de transition modélisant des distributeurs de boissons.*

Considérons de plus l'ensemble de propositions atomiques $\mathit{Prop} = \{\text{paye}, \text{soda}, \text{café}\}$. Bien que les systèmes de transition soient différents, il est clair qu'ils admettent les mêmes traces. Ils vérifient donc les mêmes propriétés linéaires. •

Déterminer si deux systèmes de transition admettent le même ensemble de traces est un problème de décision PSPACE-complet (voir la section 4.1.4 pour la définition d'un problème de décision PSPACE-complet). On peut cependant garantir que deux systèmes de transition admettent le même ensemble de traces en prouvant qu'ils sont équivalents par la relation de bisimulation.

Définition 2.8. Soient $ST_1 = (S_1, \mathit{Act}_1, \rightarrow_1, I_1, \mathit{Prop}, L_1)$ et $ST_2 = (S_2, \mathit{Act}_2, \rightarrow_2, I_2, \mathit{Prop}, L_2)$ deux systèmes de transition sur un même ensemble de propositions atomiques Prop. On dit que ST_1 et ST_2 sont équivalents par la relation de bisimulation, et on note $ST_1 \sim ST_2$, s'il existe une relation binaire $\mathcal{R} \subseteq S_1 \times S_2$ telle que :

- pour tout $s_1 \in S_1$, il existe $s_2 \in S_2$ tel que $s_1 \mathcal{R} s_2$;
- pour tout $s_2 \in S_2$, il existe $s_1 \in S_1$ tel que $s_1 \mathcal{R} s_2$;
- pour tout couple $(s_1, s_2) \in S_1 \times S_2$ tel que $s_1 \mathcal{R} s_2$, on a :
  - $L_1(s_1) = L_2(s_2)$,
  - si $s'_1 \in \mathit{Post}(s_1)$, alors il existe $s'_2 \in \mathit{Post}(s_2)$ tel que $s'_1 \mathcal{R} s'_2$,
  - si $s'_2 \in \mathit{Post}(s_2)$, alors il existe $s'_1 \in \mathit{Post}(s_1)$ tel que $s'_1 \mathcal{R} s'_2$.

□

On montre facilement que la relation de bisimulation est une relation d'équivalence. De plus, on montre que si $ST_1 \sim ST_2$, alors $\mathit{Traces}(ST_1) = \mathit{Traces}(ST_2)$. Par conséquent, si deux systèmes de transition sont équivalents par la relation de bisimulation, alors ils satisfont les mêmes propriétés linéaires. Enfin, on peut déterminer par un procédé algorithmique si deux systèmes de transition sont équivalents par la relation de bisimulation, avec une complexité en temps de l'ordre

$$\mathcal{O}\Big(\big(|S_1| + |S_2| \big) \times |\mathit{Prop}| + (M_1 + M_2) \times \log \big(|S_1| + |S_2| \big) \Big),$$

où M_1, M_2 désignent les nombres d'arêtes dans ST_1 et ST_2 respectivement.

2.2 Invariants

Les invariants sont des propriétés linéaires particulières, qui sont déterminées par une condition Φ portant sur les états d'un système de transition, et qui requiert que cette condition soit vérifiée pour les états accessibles.

Définition 2.9 (Invariant)**.** Soit *Prop* un ensemble de propositions atomiques. Une propriété linéaire P_{inv} sur *Prop* est un *invariant* s'il existe une proposition logique Φ sur *Prop* telle que

$$P_{inv} = \{A_0 A_1 A_2 \cdots \in (\, 2^{Prop}\,)^\omega \mid \forall j \geq 0, A_j \models \Phi\}\,.$$

On dit que Φ est une *condition invariante* (ou un *état invariant*) de P_{inv}. □

On remarque que

$$\begin{aligned} ST \models P_{inv} &\Leftrightarrow \mathit{trace}(\pi) \in P_{inv} \text{ pour tout chemin } \pi \text{ de } ST \\ &\Leftrightarrow L(s) \models \Phi \text{ pour tout état } s \text{ appartenant à un chemin de } ST \\ &\Leftrightarrow L(s) \models \Phi \text{ pour tout état } s \in \mathit{Access}(ST). \end{aligned}$$

Ainsi, Φ doit être vérifiée par tous les états initiaux et la satisfaction de Φ est invariante le long des fragments atteignant un état accessible. Si Φ est vérifiée pour un état s impliqué dans une transition $s \xrightarrow{\alpha} s'$, alors Φ est vérifiée pour s' également.

Exemple 2.4. La propriété d'exclusion mutuelle de deux processus concurrents (toujours au plus un processus en état critique), présentée dans l'exemple 1.7, peut être décrite comme un invariant, avec

$$\Phi = \neg \mathit{crit}_1 \vee \neg \mathit{crit}_2.$$

•

Vérification d'un invariant

La vérification d'un invariant P_{inv} pour un système de transition donné revient à vérifier la validité d'une condition invariante Φ pour chaque état accessible depuis un état initial. On peut effectuer cette vérification en adaptant un algorithme de balayage de graphe en profondeur. On obtient l'algorithme 2. Dans cet algorithme, ε désigne la pile vide ; l'instruction $\mathit{push}(s, U)$ insère s au sommet de U ; l'instruction $\mathit{top}(U)$ renvoie le premier élément de U (qui se trouve au sommet) ; l'instruction $\mathit{pop}(U)$ efface le premier élément de U.

Si au moins un sommet s visité ne satisfait pas Φ, alors l'invariance de P_{inv} n'est pas vérifiée. L'ensemble d'états R stocke tous les états visités ; si l'algorithme renvoie « OUI », alors on a $R = \mathit{Access}(ST)$. La pile U organise tous les états qui restent à visiter, pourvu qu'ils ne soient pas contenus dans R. Lorsqu'un état qui ne satisfait pas Φ est rencontré, le contenu de la pile, lue de bas en haut, fournit un contre-exemple sous la forme d'un fragment de chemin.

La complexité de l'algorithme 2 est donnée par le théorème suivant.

Théorème 2.2 (Complexité temporelle de la vérification d'invariant)**.** La complexité temporelle de l'algorithme 2 est de l'ordre de $\mathcal{O}\big(N \times (1 + |\Phi|) + M\big)$, où N désigne le nombre d'états accessibles et $M = \sum_{s \in S} |\mathit{Post}(s)|$.

Démonstration. On a rappelé précédemment que la complexité temporelle de l'algorithme d'analyse d'accessibilité est de l'ordre de $\mathcal{O}(N + M)$. De plus, le temps nécessaire pour vérifier si $s \models \Phi$ pour un état s donné est une fonction linéaire de la longueur de Φ. Pour couvrir le cas où Φ est une proposition atomique (auquel cas $|\Phi| = 0$), on ajoute 1 opération. Enfin, comme on vérifie Φ pour chaque état, le nombre d'opérations est de l'ordre de $\mathcal{O}\big((N + M) + N(1 + |\Phi|)\big)$. □

Algorithme 2 Vérification d'invariant

Entrée : système de transition fini ST et proposition logique Φ*.*
Sortie : « *OUI* » *si ST satisfait toujours* Φ*, autrement* « *NON* » *et un contre-exemple.*

```
Ensemble d'états R ← ∅                                  // l'ensemble des états accessibles
Pile d'états U ← ε                                      // ε désigne la pile vide
Booléen b ← VRAI                                        // tous les états de R vérifient Φ
Tant que I \ R ≠ ∅ ∧ b faire
    choisir s ∈ I \ R        // on choisit arbitrairement un état initial qui n'est pas dans R
    visiter(s)                                          // on appelle la procédure de balayage
Fin Tant que
Si b alors
    renvoyer « OUI »                                    // ST satisfait toujours Φ
Sinon
    renvoyer (« NON », U)                               // la pile U fournit un contre-exemple
Fin Si
```

```
Procédure visiter(état s)
    push(s, U)                                          // on pose s sur la pile
    R ← R ∪ {s}                                         // on marque s comme accessible
    Répéter
        s' ← top(U)                                     // s' est le premier élément de la pile
        Si Post(s') ⊆ R alors
            pop(U)                                      // on retire le premier élément de la pile
            b ← b ∧ (s' ⊨ Φ)                            // on vérifie la validité de Φ en s'
        Sinon
            choisir s'' ∈ Post(s') \ R
            push(s'', U)
            R ← R ∪ {s''}                               // s'' est un nouvel état accessible
        Fin Si
    Jusqu'à (U = ε) ∨ ¬b
Fin Procédure
```

2.3 Propriétés de sûreté

Les propriétés de sûreté (*safety* en anglais) peuvent être décrites par l'expression « *rien de mauvais ne devrait arriver* ». Par exemple, la propriété d'exclusion mutuelle (toujours au plus un processus en état critique) est une propriété typique de sûreté : la situation indésirable correspondant à au moins deux processus en état critique simultanément ne se produit jamais.

Contrairement aux invariants, qui peuvent être vérifiés en examinant les états accessibles d'un système de transition, certaines propriétés de sûreté ne peuvent pas être vérifiées en examinant seulement les états accessibles, et imposent des conditions supplémentaires sur certains chemins finis.

Formellement, une propriété de sûreté P_s est définie comme une propriété linéaire sur un ensemble de propositions atomiques, telle que tout mot infini σ qui ne satisfait pas P_s contient un *mauvais préfixe* : il existe alors un préfixe (fini) $\hat{\sigma}$ « mauvais », tel qu'aucun mot infini qui commence par $\hat{\sigma}$ ne vérifie P_s.

2.3.1 Mauvais préfixes

Définition 2.10 (Propriété de sûreté)**.** Une propriété linéaire P_s sur un ensemble de propositions atomiques *Prop* est appelée *propriété de sûreté* si pour chaque mot $\sigma \in \left(2^{Prop}\right)^\omega \setminus P_s$, il existe un préfixe fini $\hat{\sigma}$ de σ tel que

$$P_s \cap \left\{\sigma' \in \left(2^{Prop}\right)^\omega \mid \hat{\sigma} \text{ est un préfixe fini de } \sigma'\right\} = \varnothing.$$

Chaque mot fini $\hat{\sigma}$ est appelé *mauvais préfixe* de P_s. Un mauvais préfixe $\hat{\sigma}$ est *minimal* si aucun préfixe strict de $\hat{\sigma}$ n'est un mauvais préfixe de P_s. □

Ainsi, si $\sigma \not\models P_s$, alors σ peut s'écrire $\sigma = \hat{\sigma}\hat{w}$, où $\hat{\sigma}$ un est préfixe fini tel que pour tout mot infini w, $\hat{\sigma}w \not\models P_s$.

Pour $\sigma \in \left(2^{Prop}\right)^\omega$, on note $pref(\sigma)$ l'ensemble des préfixes finis de σ. Ainsi, si $\sigma = A_0A_1A_2\ldots$, alors

$$pref(\sigma) = \{\varepsilon, A_0, A_0A_1, A_0A_1A_2, \ldots\}.$$

Par exemple, si $\sigma = ABABABAB\ldots$, alors

$$pref(\sigma) = \{\varepsilon, A, AB, ABA, ABAB, \ldots\},$$

que l'on peut décrire avec l'expression régulière [1] $(AB)^*(A + \varepsilon)$. Enfin, si P est une propriété linéaire sur *Prop*, on note

$$pref(P) = \bigcup_{\sigma \in P} pref(\sigma).$$

L'ensemble des mauvais préfixes de P_s sera noté $MauvPre(P_s)$. L'ensemble des mauvais préfixes minimaux de P_s sera noté $MauvPreMin(P_s)$.

Définition 2.11 (Fermeture d'une propriété linéaire)**.** Soit P une propriété linéaire sur *Prop*. La *fermeture* de P est l'ensemble défini par

$$fermeture(P) = \left\{\sigma \in \left(2^{Prop}\right)^\omega \mid pref(\sigma) \subseteq pref(P)\right\}.$$

□

Ainsi, les traces infinies contenues dans $fermeture(P)$ n'ont pas de préfixe qui ne soit pas déjà un préfixe de P.

Pour toute propriété linéaire P, on a $P \subseteq fermeture(P)$. Le lemme suivant montre que l'inclusion réciproque est vérifiée pour les propriétés de sûreté.

Lemme 2.1. Soit P une propriété linéaire sur *Prop*. Alors P est une propriété de sûreté si et seulement si

$$P = fermeture(P).$$

Démonstration. Supposons d'abord que $P = fermeture(P)$. Soit alors $\sigma \in \left(2^{Prop}\right)^\omega \setminus P$. Montrons que σ commence avec un mauvais préfixe pour P. Puisque $\sigma \notin P$ et que $P = fermeture(P)$, on a donc $\sigma \notin fermeture(P)$. Donc $pref(\sigma) \not\subseteq pref(P)$; ainsi, il existe un préfixe fini $\hat{\sigma}$ de σ tel que $\hat{\sigma} \notin pref(P)$, c'est-à-dire

$$\forall \sigma \in P, \quad \hat{\sigma} \notin pref(\sigma).$$

Ainsi, si $\sigma' \in \left(2^{Prop}\right)^\omega$ est tel que $\hat{\sigma} \in pref(\sigma')$, alors $\sigma' \notin P$. Autrement dit, aucun mot $\sigma' \in \left(2^{Prop}\right)^\omega$ tel que $\hat{\sigma} \in pref(\sigma')$ n'appartient à P. Donc $\hat{\sigma}$ est un mauvais préfixe pour P. Donc P est une propriété de sûreté.

Réciproquement, supposons que P soit une propriété de sûreté. L'inclusion $P \subseteq fermeture(P)$ est évidente. Pour montrer l'autre inclusion, on raisonne par contradiction. Supposons que $\sigma = A_0A_1A_2\ldots$

1. Les expressions régulières seront revues dans le troisième chapitre de ce cours.

appartienne à $\mathit{fermeture}(P) \setminus P$. Comme P est une propriété de sûreté et que $\sigma \notin P$, σ admet un mauvais préfixe $\hat{\sigma} = A_0 A_1 \dots A_n$. Mais $\sigma \in \mathit{fermeture}(P)$, donc $\mathit{pref}(\sigma) \subseteq \mathit{pref}(P)$, d'où $\hat{\sigma} \in \mathit{pref}(P)$. Donc il existe un mot $\sigma' \in P$ admettant $\hat{\sigma}$ comme préfixe, ce qui est impossible puisque P est une propriété de sûreté. □

Le théorème suivant établit une relation entre invariants et propriétés de sûreté.

Théorème 2.3 (Relation entre invariants et propriétés de sûreté)**.** Tout invariant est une propriété de sûreté.

Démonstration. Soit P_{inv} un invariant et soit Φ une condition invariante de P_{inv}. Soit ensuite σ un mot infini de $\left(2^{Prop}\right)^{\omega} \setminus P_{inv}$. Alors σ contient un préfixe minimal du type

$$A_0 A_1 \dots A_n \in \left(2^{Prop}\right)^{+} \text{ avec } A_0 \models \Phi, \dots, A_{n-1} \models \Phi \text{ et } A_n \not\models \Phi. \tag{2.1}$$

Donc P_{inv} est une propriété de sûreté et tous les mots finis de la forme (2.1) en constituent les mauvais préfixes minimaux. □

Exemple 2.5. Considérons un distributeur de boissons. La condition suivante est une condition requise naturelle :

« *le nombre de pièces insérées est toujours au moins égal au nombre de boissons distribuées* ».

Pour formaliser cette condition, on introduit l'ensemble de propositions atomiques

$$\mathit{Prop} = \{\text{paye}, \text{distribue}\},$$

et la fonction d'étiquetage associée. La condition énoncée ci-dessus correspond à l'ensemble des mots infinis $A_0 A_1 A_2 \dots$ tels que l'on ait pour tout $i \geq 0$:

$$\left|\left\{j \in \{0, \dots, i\} \mid \text{paye} \in A_j\right\}\right| \geq \left|\left\{j \in \{0, \dots, i\} \mid \text{distribue} \in A_j\right\}\right|.$$

Cet ensemble de mots constitue alors une propriété de sûreté, dont les mots suivants sont des exemples de mauvais préfixes :

$$\varnothing\{\text{paye}\}\{\text{distribue}\}\{\text{distribue}\},$$
$$\varnothing\{\text{paye}\}\{\text{distribue}\}\varnothing\{\text{paye}\}\{\text{distribue}\}\{\text{distribue}\}.$$

On vérifie aisément que les distributeurs de boissons de la figure 2.3 satisfont cette propriété de sûreté. •

2.3.2 Équivalence de traces pour les propriétés de sûreté

Nous avons vu avec le Théorème 2.1 qu'il existe une relation entre l'inclusion de traces pour des systèmes de transition sans état final et la satisfaction de propriétés linéaires. Le Théorème 2.1 s'applique néanmoins à des traces infinies. Une relation analogue pour des traces finies est établie par le théorème suivant.

Théorème 2.4 (Inclusion de traces finies et propriétés de sûreté)**.** Soient ST et ST' deux systèmes de transition sans état final et admettant le même ensemble de propositions atomiques. Les assertions suivantes sont équivalentes :

- $\mathit{Traces}_{\mathit{fin}}(ST) \subseteq \mathit{Traces}_{\mathit{fin}}(ST')$;
- pour toute propriété de sûreté P_s, $ST' \models P_s$ implique $ST \models P_s$.

La démonstration du Théorème 2.4 utilise le lemme suivant.

Lemme 2.2. Soit ST un système de transition sans état final et soit P_s une propriété de sûreté. Alors

$$ST \models P_s \Leftrightarrow \mathit{Traces}_{\mathit{fin}}(ST) \cap \mathit{MauvPre}(P_s) = \varnothing.$$

Preuve du Lemme 2.2. (1) $\Rightarrow$ (2). Raisonnons par l'absurde. Supposons que $ST \models P_s$ et qu'il existe $\hat{\sigma} \in \mathit{Traces}_{\mathit{fin}}(ST) \cap \mathit{MauvPre}(P_s)$. Alors $\hat{\sigma}$ est une trace finie de ST qui peut s'écrire

$$\hat{\sigma} = A_1 \dots A_n,$$

et qui peut être prolongée en une trace infinie

$$\sigma = A_1 \dots A_n A_{n+1} \dots$$

qui ne satisfait pas P_s. Donc $ST \not\models P_s$.

(2) $\Rightarrow$ (1). Raisonnons encore par l'absurde. Supposons que $\mathit{Traces}_{\mathit{fin}}(ST) \cap \mathit{MauvPre}(P_s) = \varnothing$ et que $ST \not\models P_s$. Alors il existe un chemin π de ST tel que $\mathit{trace}(\pi) \notin P_s$. Cela signifie que $\mathit{trace}(\pi)$ commence avec un mauvais préfixe $\hat{\sigma}$ pour P_s. Mais on a alors $\hat{\sigma} \in \mathit{Traces}_{\mathit{fin}}(ST) \cap \mathit{MauvPre}(P_s)$, ce qui constitue une contradiction. □

Preuve du Théorème 2.4. (1) $\Rightarrow$ (2). Supposons que $\mathit{Traces}_{\mathit{fin}}(ST) \subseteq \mathit{Traces}_{\mathit{fin}}(ST')$ et soit P_s une propriété de sûreté telle que $ST' \models P_s$. D'après le lemme 2.2, on a donc

$$\mathit{Traces}_{\mathit{fin}}(ST') \cap \mathit{MauvPre}(P_s) = \varnothing,$$

ce qui implique

$$\mathit{Traces}_{\mathit{fin}}(ST) \cap \mathit{MauvPre}(P_s) = \varnothing.$$

D'après le lemme 2.2, on a donc $ST \models P_s$.

(2) $\Rightarrow$ (1). Considérons $P_s = \mathit{fermeture}\big(\mathit{Traces}(ST')\big)$. On montre aisément que P_s est une propriété de sûreté telle que $ST' \models P_s$. Par hypothèse, on a donc $ST \models P_s$, c'est-à-dire

$$\mathit{Traces}(ST) \subseteq \mathit{fermeture}\big(\mathit{Traces}(ST')\big).$$

Sachant que pour toute propriété linéaire P, on a

$$\mathit{pref}(P) = \mathit{pref}\big(\mathit{fermeture}(P)\big),$$

on obtient :

$$\begin{aligned}
\mathit{Traces}_{\mathit{fin}}(ST) &= \mathit{pref}\big(\mathit{Traces}(ST)\big) \\
&\subseteq \mathit{pref}\Big(\mathit{fermeture}\big(\mathit{Traces}(ST')\big)\Big) \\
&= \mathit{pref}\big(\mathit{Traces}(ST')\big) \\
&= \mathit{Traces}_{\mathit{fin}}(ST').
\end{aligned}$$

□

On obtient le corollaire suivant.

Corollaire 2.2 (Traces finies équivalentes)**.** Soient ST et ST' deux systèmes de transition sans état final sur le même ensemble de propositions atomiques Prop. Alors les assertions suivantes sont équivalentes :

- $\mathit{Traces}_{\mathit{fin}}(ST) = \mathit{Traces}_{\mathit{fin}}(ST')$,
- pour toute propriété de sûreté P_s sur Prop, $ST \models P_s \Leftrightarrow ST' \models P_s$.

Exemple 2.6. Considérons un feu de circulation avec trois phases « rouge », « vert », « orange ». Nous allons montrer que la propriété P_s définie par

$$\ll \textit{ chaque phase “rouge” est immédiatement précédée d'une phase “orange” } \gg \tag{2.2}$$

est une propriété de sûreté, mais pas un invariant.

Pour cela, on pose $Prop = \{\mathsf{rouge}, \mathsf{orange}, \mathsf{vert}\}$. La propriété « *il y a toujours au moins une lumière allumée* » peut s'écrire

$$\{\sigma = A_0 A_1 \dots \mid \forall j \geq 0, A_j \subseteq Prop \wedge A_j \neq \varnothing\}.$$

Les mauvais préfixes de cette propriété sont les mots finis qui contiennent $\varnothing$. Un mauvais préfixe minimal finit nécessairement par $\varnothing$. Puis, la propriété « *deux lumières ne sont jamais allumées en même temps* » peut s'écrire

$$\{\sigma = A_0 A_1 \dots \mid \forall j \geq 0, A_j \subseteq Prop \wedge |A_j| \leq 1\}.$$

Les mauvais préfixes de cette propriété sont les mots finis contenant $\{\mathsf{rouge}, \mathsf{vert}\}$, $\{\mathsf{rouge}, \mathsf{orange}\}$ ou $\{\mathsf{orange}, \mathsf{vert}\}$. Un mauvais préfixe minimal finit nécessairement par un tel ensemble.

Considérons maintenant l'ensemble de propositions atomiques $Prop' = \{\mathsf{rouge}, \mathsf{orange}\}$. La propriété P_s définie par 2.2 s'écrit

$$\big\{\sigma = A_0 A_1 \dots \mid \forall j \geq 0, A_j \subseteq Prop' \wedge (\mathsf{rouge} \in A_j \Rightarrow j > 0 \wedge \mathsf{orange} \in A_{j-1})\big\}.$$

Les mauvais préfixes de cette propriété sont les mots finis qui ne vérifient pas la condition

$$(\mathsf{rouge} \in A_j \Rightarrow j > 0 \wedge \mathsf{orange} \in A_{j-1}).$$

Le mot “$\varnothing\varnothing\{\mathsf{rouge}\}$” est un mauvais préfixe minimal. Le mauvais préfixe

$$\{\mathsf{orange}\}\{\mathsf{orange}\}\{\mathsf{rouge}\}\{\mathsf{rouge}\}\varnothing\{\mathsf{rouge}\}$$

n'est pas minimal puisque qu'il admet le préfixe strict

$$\{\mathsf{orange}\}\{\mathsf{orange}\}\{\mathsf{rouge}\}\{\mathsf{rouge}\},$$

qui est aussi un mauvais préfixe.

Les mauvais préfixes minimaux de cette propriété de sûreté constituent un langage régulier. L'automate fini de la figure 2.4 accepte précisément ces mauvais préfixes minimaux (dans cet automate, l'expression « $\neg\mathsf{orange}$ » signifie « soit $\varnothing$, soit “rouge” »). •

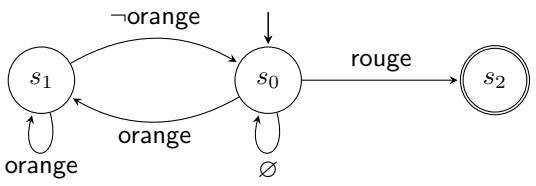

Figure 2.4. *Automate fini acceptant les mauvais préfixes minimaux pour la propriété de sûreté P_s définie par (2.2).*

2.4 Propriétés de vivacité

Pour terminer ce deuxième chapitre du cours, décrivons brièvement une autre catégorie remarquable de propriétés linéaires : les propriétés de *vivacité* (*liveness* en anglais).

Les propriétés de vivacité peuvent être décrites par l'expression « *quelque chose de bon devrait arriver dans le futur* ». Ces propriétés sont alors parfois appelées *progrès*, et sont complémentaires des propriétés de sûreté. La définition formelle suivante d'une propriété de vivacité utilise encore la notion de préfixe.

Définition 2.12 (Propriété de vivacité)**.** Une propriété linéaire P_v sur un ensemble de propositions atomiques *Prop* est appelée *propriété de vivacité* si $\mathit{pref}(P_v) = \left(2^{Prop}\right)^*$. □

Ainsi, P_v est une propriété de vivacité si chaque mot fini peut être prolongé en un mot infini qui vérifie P_v. On peut facilement montrer que la seule propriété linéaire qui soit à la fois une propriété de sûreté et une propriété de vivacité est $\left(2^{Prop}\right)^\omega$. Le théorème suivant montre l'importance conjointe des propriétés de sûreté et de vivacité.

Théorème 2.5 (Théorème de décomposition)**.** Pour toute propriété linéaire P, il existe une propriété de sûreté P_s et une propriété de vivacité P_v telles que

$$P = P_s \cap P_v.$$

Remarque. Il n'y a pas unicité de la décomposition $P = P_s \cap P_v$. Cependant, on peut montrer que la décomposition la plus « fine » est obtenue avec

$$P_s = \mathit{fermeture}(P) \quad \text{et} \quad P_v = P \cup \Big(\left(2^{Prop}\right)^\omega \setminus \mathit{fermeture}(P)\Big).$$

◁

2.5 Exercices

Exercice 1 : le blocage

On considère 2 feux de circulation disposés sur des routes qui se croisent, modélisés par les systèmes de transition FC_1 et FC_2 de la figure 2.5.

1. Décrire la composition parallèle synchrone $FC_1 \parallel FC_2$ des systèmes de transition FC_1 et FC_2.
2. Que remarque-t-on ?

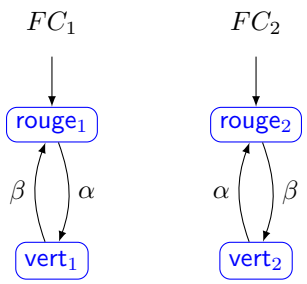

Figure 2.5. *Feux de circulation.*

Exercice 2 : le distributeur de boissons

On considère un distributeur de boissons, modélisé par le système de transition illustré sur la figure 2.6, muni de la fonction d'étiquetage définie par $L(s) = \{s\}$ pour tout état s.

1. Donner les fragments de chemins correspondant aux fragments d'exécution ρ_1, ρ_2 et ρ_3 donnés par

$$\rho_1 = \text{paye} \xrightarrow{\text{inserer\_piece}} \text{choix} \xrightarrow{\text{choix\_interne}} \text{café} \dots ,$$
$$\rho_2 = \text{café} \xrightarrow{\text{servir\_cafe}} \text{paye} \xrightarrow{\text{inserer\_piece}} \text{choix} \xrightarrow{\text{choix\_interne}} \text{café} \dots ,$$
$$\rho_3 = \text{paye} \xrightarrow{\text{inserer\_piece}} \text{choix} \xrightarrow{\text{choix\_interne}} \text{café} \xrightarrow{\text{servir\_cafe}} \text{paye} .$$

2. Décrire les propriétés de ces fragments de chemin et donner leur trace.

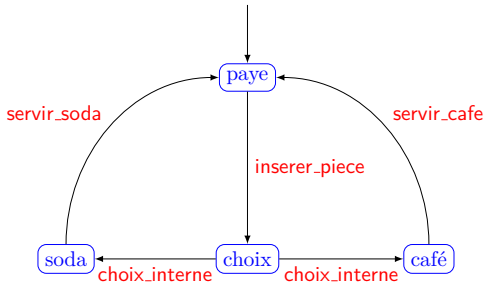

Figure 2.6. *Système de transition modélisant un distributeur de boissons.*

Exercice 3 : exclusion mutuelle par sémaphore

On considère le système de transition ST_{sem} obtenu par déploiement du graphe $P_1 \,|||\, P_2$, où les processus P_1, P_2, illustrés sur la figure 2.7, partagent le sémaphore binaire y. On suppose que l'ensemble de propositions atomiques est donné par $Prop = \{\mathsf{crit}_1, \mathsf{crit}_2\}$.

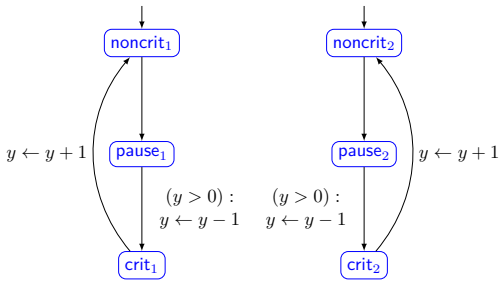

Figure 2.7. *Processus concurrents partageant le sémaphore binaire y.*

1. Décrire les fragments de chemins π et $\hat{\pi}$ définis par :

$$\pi = < \mathsf{noncrit}_1, \mathsf{noncrit}_2, y = 1 > \rightarrow < \mathsf{pause}_1, \mathsf{noncrit}_2, y = 1 > \rightarrow < \mathsf{crit}_1, \mathsf{noncrit}_2, y = 0 > \rightarrow$$
$$< \mathsf{noncrit}_1, \mathsf{noncrit}_2, y = 1 > \rightarrow < \mathsf{noncrit}_1, \mathsf{pause}_2, y = 1 > \rightarrow < \mathsf{noncrit}_1, \mathsf{crit}_2, y = 0 > \rightarrow \dots$$
$$\hat{\pi} = < \mathsf{noncrit}_1, \mathsf{noncrit}_2, y = 1 > \rightarrow < \mathsf{pause}_1, \mathsf{noncrit}_2, y = 1 > \rightarrow < \mathsf{pause}_1, \mathsf{pause}_2, y = 1 > \rightarrow$$
$$< \mathsf{pause}_1, \mathsf{crit}_2, y = 0 > \rightarrow < \mathsf{pause}_1, \mathsf{noncrit}_2, y = 1 > \rightarrow < \mathsf{crit}_1, \mathsf{noncrit}_2, y = 0 >,$$

puis donner leur trace.

2. Soit la propriété Φ_1 : « *toujours au plus un des deux processus est dans son état critique* ». Montrer que la propriété Φ_1 est une propriété linéaire. Donner des exemples de mots infinis appartenant à Φ_1, puis un mot infini n'appartenant pas à Φ_1. Montrer que Φ_1 est un invariant puis exécuter l'algorithme de vérification d'invariant pour décider si ST_{sem} satisfait Φ_1 ou non.
3. Analyser de même la propriété Φ_2 : « *toujours aucun processus n'est en état critique* ».
4. Que dire d'un algorithme qui interdit aux deux processus d'entrer dans leur état critique ?
5. On considère maintenant l'ensemble de propositions atomiques

$$Prop' = \{\mathsf{pause}_1, \mathsf{pause}_2, \mathsf{crit}_1, \mathsf{crit}_2\}.$$

   Décrire comme un ensemble de mots la propriété Φ_3 sur $Prop'$ qui exprime que chaque processus entre effectivement en état critique après un temps d'attente fini.
6. Décrire comme un ensemble de mots la propriété Φ_4 sur $Prop'$ qui exprime que chaque processus entre en état critique infiniment souvent, à condition d'attendre une infinité de fois.
7. Les propriétés Φ_3, Φ_4 sont-elles satisfaites par le système de transition ST_{sem} ?
8. Sont-elles des propriétés de sûreté ? de vivacité ?

Exercice 4

On considère un ensemble fini *Prop* de propositions atomiques.

1. Soit ST un système de transition sur *Prop*. Montrer que l'ensemble $\mathit{fermeture}\big(\mathit{Traces}(ST)\big)$ est une propriété de sûreté, qui est vérifiée par ST.
2. Soit P une propriété linéaire sur *Prop*. Montrer que $\mathit{pref}(P) = \mathit{pref}\big(\mathit{fermeture}(P)\big)$.
3. Montrer que la seule propriété linéaire qui soit à la fois une propriété de sûreté et une propriété de vivacité est $\big(2^{Prop}\big)^{\omega}$.

Exercice 5 : l'algorithme d'exclusion de Peterson

On considère deux processus P_1, P_2 possédant un état critique noté crit, précédé d'un état d'attente noté pause, lui-même précédé d'un état non critique noté $\mathsf{noncrit}$. On suppose que P_1 et P_2 admettent en commun les variables de type booléen b_1, b_2 et la variable x dont le domaine est $\mathit{dom}(x) = \{1, 2\}$. Si les deux processus désirent entrer dans leur état critique, alors la valeur de la variable x effectue la décision : si $x = i$, c'est P_i qui entre dans son état critique. De plus, si P_1 (respectivement P_2) entre en état d'attente, alors la variable x prend la valeur 2 (respectivement 1). Les variables b_i sont définies par $b_i = \mathsf{pause}_i \vee \mathsf{crit}_i$.

1. Représenter chaque processus par un graphe conditionnel et contruire le système de transition ST_{Peter} obtenu par déploiement de leur composition.
2. Le système de transition ST_{Peter} vérifie-t-il la propriété d'exclusion mutuelle ? Quelle est sa particularité ?
3. Le système de transition ST_{Peter} vérifie-t-il les propriétés Φ_3, Φ_4 de l'exercice 3 ?

CHAPITRE 3

Vérification de propriétés de sûreté

Nous avons vu dans le chapitre précédent que la vérification des invariants peut être effectuée en appliquant un algorithme d'analyse d'accessibilité dans le graphe d'états d'un système de transition. Dans ce chapitre, nous allons présenter une méthode permettant de vérifier d'autres propriétés linéaires. Nous considérons principalement des propriétés de sûreté *régulières*, c'est-à-dire des propriétés de sûreté dont les mauvais préfixes constituent un langage régulier, qui peut donc être reconnu par un automate fini. Nous allons alors décrire un algorithme de vérification d'une telle propriété de sûreté P_s pour un système de transition ST, qui repose sur une réduction à un problème de vérification d'invariant, pour un certain produit $ST \otimes \mathcal{A}$, où ST est le système de transition considéré et $\mathcal{A}$ un automate fini qui reconnaît les mauvais préfixes de P_s.

3.1 Automates finis et langages réguliers

Nous commençons par présenter quelques rappels sur les automates finis.

3.1.1 Automate fini non-déterministe

Définition 3.1 (Automate Fini Non-déterministe)**.** Un *automate fini non-déterministe* $\mathcal{A}$ est entièrement déterminé par la donnée d'un quintuplet $\mathcal{A} = (Q, \Sigma, \delta, Q_0, F)$ où

- Q est un ensemble fini d'états,
- Σ est un alphabet,
- $\delta : Q \times \Sigma \longrightarrow 2^Q$ est une fonction de transition,
- $Q_0 \subseteq Q$ est un ensemble d'états initiaux,
- $F \subseteq Q$ est un ensemble d'états *acceptants* (ou finaux).

La taille de $\mathcal{A}$, notée $|\mathcal{A}|$, est le nombre d'états et de transitions dans $\mathcal{A}$:

$$|\mathcal{A}| = |Q| + \sum_{q \in Q} \sum_{A \in \Sigma} |\delta(q, A)| \,.$$

□

L'alphabet Σ détermine ainsi les symboles sur lesquels l'automate $\mathcal{A}$ est défini. L'ensemble Q_0 (qui peut être vide) détermine les états dans lesquels l'automate $\mathcal{A}$ peut débuter. La fonction de transition

δ peut être identifiée avec la relation $\longrightarrow \subseteq Q \times \Sigma \times Q$ définie par

$$q \xrightarrow{A} q' \Leftrightarrow q' \in \delta(q, A).$$

Intuitivement, la notation $q \xrightarrow{A} q'$ décrit l'automate $\mathcal{A}$ qui se déplace de l'état q à l'état q' lorsqu'il lit le symbole A.

Exemple 3.1. Considérons l'automate illustré sur la figure 3.1. Dans cet exemple, on a $Q = \{q_0, q_1, q_2\}$, $\Sigma = \{A, B\}$, $Q_0 = \{q_0\}$, $F = \{q_2\}$ et la fonction de transition δ est définie par

$$\begin{array}{ll} \delta(q_0, A) = \{q_0\}, & \delta(q_0, B) = \{q_0, q_1\}, \\ \delta(q_1, A) = \{q_2\}, & \delta(q_1, B) = \{q_2\}, \\ \delta(q_2, A) = \varnothing, & \delta(q_2, B) = \varnothing. \end{array}$$

Les conventions de dessin pour les automates sont les mêmes que pour les systèmes de transition. Les états acceptants sont distingués des autres par un double cercle.

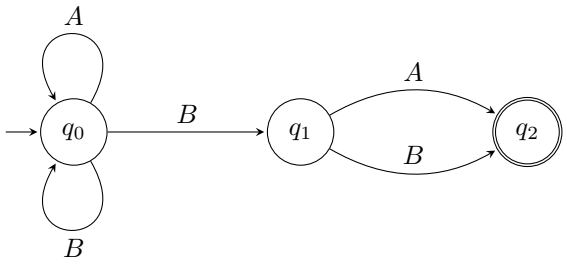

Figure 3.1. *Exemple d'automate fini non-déterministe.*

•

3.1.2 Langage accepté par un automate

Le comportement opérationnel d'un automate peut être décrit de la façon suivante. L'automate débute dans un état $q_0 \in Q_0$. On lui donne un mot $w \in \Sigma^*$ à lire. L'automate lit alors ce mot w caractère par caractère, de gauche à droite. Après lecture d'un caractère, l'automate change d'état selon la fonction de transition δ. Si le symbole A lu à partir de l'état q est tel que $\delta(q, A)$ contient plus d'un état (ce qui se produit pour $\delta(q_0, B)$ dans l'exemple 3.1), alors la décision pour l'état suivant est prise de façon non déterministe. Si $\delta(q, A) = \varnothing$, alors l'automate est bloqué. Si l'automate est bloqué avant la fin du mot w, on dit que le mot w est *rejeté*. Si au contraire le mot w est lu de façon complète, l'automate s'arrête. On dit qu'il *accepte* le mot w s'il s'arrête dans un état acceptant. Autrement, on dit qu'il *rejette* le mot w.

Définition 3.2 (Langage accepté par un automate)**.** Soit $\mathcal{A} = (Q, \Sigma, \delta, Q_0, F)$ un automate fini non-déterministe et soit $w = A_1 \dots A_n \in \Sigma^*$ un mot fini. On appelle *exécution* de l'automate $\mathcal{A}$ pour le mot w toute suite finie d'états $q_0 q_1 \dots q_n$ telle que

- $q_0 \in Q_0$,
- $q_i \xrightarrow{A_{i+1}} q_{i+1}$ pour tout $i \in \{0, \dots, n-1\}$.

On dit que l'exécution $q_0 q_1 \dots q_n$ est *acceptante* si $q_n \in F$. Un mot $w \in \Sigma^*$ est dit *accepté* par l'automate $\mathcal{A}$ s'il existe au moins une exécution acceptante pour w.

Le *langage accepté* par $\mathcal{A}$, noté $\mathcal{L}(\mathcal{A})$, est l'ensemble des mots finis dans Σ^* qui sont acceptés par $\mathcal{A}$. □

Exemple 3.2. Pour l'automate illustré sur la figure 3.1, q_0 est une exécution pour le mot vide ε, $q_0q_0q_0q_0$ est une exécution pour les mots ABA et BBA et $q_0q_1q_2$ est une exécution pour les mots BA et BB. Les exécutions $q_0q_1q_2$ pour BA et BB et $q_0q_0q_1q_2$ pour ABB, ABA, BBA et BBB sont acceptantes. Donc ces mots appartiennent au langage $\mathcal{L}(\mathcal{A})$. En revanche, le mot AAA n'est pas accepté par $\mathcal{A}$, car il n'admet qu'une seule exécution, $q_0q_0q_0q_0$, qui n'est pas acceptante. Le langage $\mathcal{L}(\mathcal{A})$ est ainsi déterminé par l'expression régulière $(A+B)^*B(A+B)$. Autrement dit, $\mathcal{L}(\mathcal{A})$ est l'ensemble des mots sur $\{A,B\}$ admettant au moins 2 lettres, et dont l'avant-dernière lettre est B. •

Les cas $Q_0 = \varnothing$ et $F = \varnothing$ sont possibles. Dans ces 2 cas, on a $\mathcal{L}(\mathcal{A}) = \varnothing$. Si $F = \varnothing$, alors il n'existe aucune exécution acceptante. Si $Q_0 = \varnothing$, alors il n'existe aucune exécution (intuitivement, l'automate rejette d'emblée tout mot à lire).

3.1.3 Expressions régulières

Soit Σ un alphabet non vide. Les expressions régulières[1] sont construites à partir des symboles $\underline{\varnothing}$ (pour désigner le langage vide, qui ne contient aucun mot), $\underline{\varepsilon}$ (pour désigner le langage neutre $\{\varepsilon\}$), $\underline{A}$, avec $A \in \Sigma$ (pour le singleton $\{A\}$), et des opérateurs de langages : $\ll+\gg$ pour l'union, $\ll*\gg$ pour la répétition finie (étoile de Kleene) et $\ll\cdot\gg$ pour la concaténation. Les expressions régulières sont définies de façon inductive par les règles suivantes :

- $\underline{\varnothing}$ et $\underline{\varepsilon}$ sont des expressions régulières ;
- si $A \in \Sigma$, alors $\underline{A}$ est une expression régulière ;
- si E, E_1 et E_2 sont des expressions régulières, alors $E_1 + E_2$, $E_1 \cdot E_2$ et E_1^* sont aussi des expressions régulières ;
- rien d'autre n'est une expression régulière.

Si E est une expression régulière, on définit son langage $\mathcal{L}(E)$ à partir des règles de sémantique suivantes :

$$\mathcal{L}\big(\underline{\varnothing}\big) = \varnothing, \quad \mathcal{L}\big(\underline{\varepsilon}\big) = \{\varepsilon\}, \quad \mathcal{L}\big(\underline{A}\big) = \{A\},$$
$$\mathcal{L}(E_1 + E_2) = \mathcal{L}(E_1) \cup \mathcal{L}(E_2), \quad \mathcal{L}(E_1 \cdot E_2) = \mathcal{L}(E_1) \cdot \mathcal{L}(E_2), \quad \mathcal{L}\big(E^*\big) = \mathcal{L}(E)^*.$$

On note souvent E^+ pour désigner l'expression régulière $E \cdot E^*$. On montre alors que $\mathcal{L}\big(E^+\big) = \mathcal{L}(E)^+$.

Un langage $\mathcal{L} \subseteq \Sigma^*$ est dit *régulier* s'il existe une expression régulière E sur Σ telle que $\mathcal{L}(E) = \mathcal{L}$. Par exemple, l'expression $E = \big(\underline{A} + \underline{B}\big)^* \cdot \underline{B} \cdot \underline{B} \cdot \big(\underline{A} + \underline{B}\big)$ est une expression régulière sur l'alphabet $\Sigma = \{A, B\}$, qui correspond au langage

$$\mathcal{L}(E) = \{wBBA \mid w \in \Sigma^*\} \cup \{wB^3 \mid w \in \Sigma^*\}.$$

On allège très souvent les notations en omettant la barre inférieure et le point de concaténation.

Le théorème de Kleene [9] montre que l'ensemble des langages réguliers sur un alphabet Σ est exactement l'ensemble des langages sur Σ reconnaissables par automate fini (voir la section 3.1.5 plus bas). Étant donné un automate fini non-déterministe $\mathcal{A}$, il existe même des algorithmes qui déterminent une expression régulière E telle que $\mathcal{L}(E) = \mathcal{L}(\mathcal{A})$. Réciproquement, pour toute expression régulière E, on peut construire un automate fini non-déterministe $\mathcal{A}$ qui accepte $\mathcal{L}(E)$.

Un exemple de langage non régulier est donné par $\mathcal{L} = \{A^nB^n \mid n \geq 0\}$.

3.1.4 Propriétés remarquables

On peut caractériser le langage accepté par un automate fini non-déterministe de la façon suivante. Soit $\mathcal{A}$ un tel automate. On étend la fonction de transition δ à $Q \times \Sigma^*$ en posant

$$\delta(q, \varepsilon) = \{q\}, \quad \delta(q, A_1A_2 \dots A_n) = \bigcup_{p \in \delta(q, A_1)} \delta(p, A_2 \dots A_n).$$

1. *Regular expressions* en anglais, parfois traduit *expressions rationnelles* en français.

Autrement dit, $\delta(q, w)$ est l'ensemble des états qui sont accessibles depuis l'état q avec le mot w. En particulier, $\cup_{q_0 \in Q_0} \delta(q_0, w)$ est l'ensemble des états où une exécution pour w peut terminer. Si un de ces états est acceptant, alors le mot w admet une exécution acceptante. Inversement, si $w \notin \mathcal{L}(\mathcal{A})$, alors aucun de ces états n'est acceptant. On obtient donc la caractérisation suivante.

Lemme 3.1 (Caractérisation du langage accepté par un automate fini non-déterministe)**.** Soit $\mathcal{A}$ un automate fini non-déterministe. Alors

$$\mathcal{L}(\mathcal{A}) = \big\{ w \in \Sigma^* \mid \exists\, q_0 \in Q_0 \text{ tel que } \delta(q_0, w) \cap F \neq \varnothing \big\}.$$

Puisque les automates finis non-déterministes correspondent aux langages réguliers, on identifie naturellement ceux qui admettent le même langage.

Définition 3.3 (Automates finis non-déterministes équivalents)**.** Deux automates finis non-déterministes sont dits *équivalents* s'ils admettent le même langage accepté. □

Non vacuité du langage d'un automate

Une question importante dans l'étude d'un automate consiste à déterminer si son langage accepté est vide ou non.

Théorème 3.1. Soit $\mathcal{A} = (Q, \Sigma, \delta, Q_0, F)$ un automate fini non-déterministe. Alors $\mathcal{L}(\mathcal{A}) \neq \varnothing$ si et seulement si il existe $q_0 \in Q_0$ et $q \in F$ tels que q soit accessible depuis q_0.

Démonstration. Supposons qu'il existe $q_0 \in Q_0$ et $q \in F$ tels que q soit accessible depuis q_0. Alors $\mathcal{A}$ accepte le mot correspondant à cette exécution acceptante, donc $\mathcal{L}(\mathcal{A}) \neq \varnothing$.

Réciproquement, si $\mathcal{L}(\mathcal{A}) \neq \varnothing$, alors il existe $w \in \mathcal{L}(\mathcal{A})$ et une exécution acceptante pour w, qui commence en $q_0 \in Q_0$ et qui termine en $q \in F$. □

En adaptant un algorithme de balayage, on peut décider si $\mathcal{L}(\mathcal{A})$ est vide, avec un temps de l'ordre de $O(|\mathcal{A}|)$.

Les langages réguliers vérifient de nombreuses propriétés de compatibilité avec les opérations sur les langages. Ainsi, l'union de deux langages réguliers est un langage régulier. De même pour la concaténation, l'intersection, le complémentaire et la répétition finie. On peut montrer ces propriétés en utilisant la caractérisation des langages réguliers par des automates finis non-déterministes. En particulier, la propriété de compatibilité avec l'intersection peut être établie en construisant, à partir de deux automates $\mathcal{A}_1$ et $\mathcal{A}_2$, un automate produit $\mathcal{A}_1 \otimes \mathcal{A}_2$. Cette construction est analogue à celle de l'opérateur d'échange synchrone $\|$ de deux systèmes de transition (Définition 1.13).

Définition 3.4 (Produit synchrone d'automates finis non-déterministes)**.** Soient deux automates finis non-déterministes $\mathcal{A}_1 = (Q_1, \Sigma, \delta_1, Q_{0,1}, F_1)$ et $\mathcal{A}_2 = (Q_2, \Sigma, \delta_2, Q_{0,2}, F_2)$ sur un même alphabet Σ. L'automate produit $\mathcal{A}_1 \otimes \mathcal{A}_2$ est défini par

$$\mathcal{A}_1 \otimes \mathcal{A}_2 = (Q_1 \times Q_2, \Sigma, \delta, Q_{0,1} \times Q_{0,2}, F_1 \times F_2),$$

où la fonction de transition δ est donnée par

$$\frac{q_1 \xrightarrow{A}_1 q_1' \ \wedge \ q_2 \xrightarrow{A}_2 q_2'}{(q_1, q_2) \xrightarrow{A} (q_1', q_2')}.$$

□

On montre alors que $\mathcal{L}(\mathcal{A}_1 \otimes \mathcal{A}_2) = \mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2)$.

Pour montrer la propriété de compatibilité par complémentaire, on introduit d'abord la notion d'automate fini déterministe.

Définition 3.5 (Automate fini déterministe)**.** Soit $\mathcal{A} = (Q, \Sigma, \delta, Q_0, F)$ un automate fini non-déterministe. On dit que $\mathcal{A}$ est *déterministe* si $|Q_0| \leq 1$ et $|\delta(q, A)| \leq 1$ pour tout état $q \in Q$ et pour tout symbole $A \in \Sigma$. On dit que $\mathcal{A}$ est *complet* si $|Q_0| = 1$ et $|\delta(q, A)| = 1$ pour tout état $q \in Q$ et pour tout symbole $A \in \Sigma$. □

Un automate fini complet est souvent noté $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$, où q_0 est l'unique état initial. On observe qu'un automate complet admet une unique exécution pour chaque mot donné en lecture. On définit alors, pour un automate complet $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$, l'automate $\bar{\mathcal{A}} = (Q, \Sigma, \delta, q_0, Q \setminus F)$. On montre ensuite que $\mathcal{L}(\bar{\mathcal{A}}) = \Sigma^* \setminus \mathcal{L}(\mathcal{A})$.

Il reste enfin à montrer que pour chaque automate fini non déterministe $\mathcal{A}$, il existe un automate fini complet $\mathcal{A}_c$ équivalent. Pour cela, on pose

$$\mathcal{A}_c = \big(2^Q, \Sigma, \delta_c, Q_0, F_c\big),$$

où l'ensemble des états acceptants F_c est défini par

$$F_c = \{Q' \subseteq Q \mid Q' \cap F \neq \varnothing\}$$

et où la fonction de transition δ_c est définie sur $2^Q \times \Sigma$ par

$$\delta_c(Q', A) = \bigcup_{q \in Q'} \delta(q, A).$$

La construction précédente est appelée *powerset construction* ou *subset construction*, car les états de $\mathcal{A}_c$ sont des sous-ensembles de Q. Ainsi construit, l'automate $\mathcal{A}_c$ est complet et on a

$$\delta_c(Q_0, w) = \bigcup_{q_0 \in Q_0} \delta(q_0, w),$$

pour chaque mot $w \in \Sigma^*$ (δ désigne la fonction de transition étendue). On a donc, d'après le Lemme 3.1, $\mathcal{L}(\mathcal{A}_c) = \mathcal{L}(\mathcal{A})$.

Exemple 3.3. L'automate complet obtenu par la méthode *powerset construction* à partir de l'automate fini non-déterministe de la figure 3.1 est donné sur la figure 3.2.

•

La méthode *powerset construction* produit un automate dont la taille est exponentielle par rapport à la taille de l'automate de départ. Par exemple, le langage régulier déterminé par l'expression régulière $E_k = (A+B)^*B(A+B)^k$ est accepté par un automate fini non-déterministe à $k+2$ états ; cependant, on peut montrer qu'il n'existe pas d'automate complet à moins de 2^k états acceptant le langage $\mathcal{L}(E_k)$. En effet, intuitivement, un automate complet acceptant le langage $\mathcal{L}(E_k)$ doit « se souvenir » des positions du symbole B parmi les k derniers caractères lus, ce qui produit au moins $\Omega(2^k)$ états.

Enfin, on peut montrer que pour tout langage régulier $\mathcal{L}$, il existe un unique automate complet $\mathcal{A}$ tel que $\mathcal{L}(\mathcal{A}) = \mathcal{L}$ et admettant un nombre minimal d'états, à isomorphisme de renommage des états près.

3.1.5 Le théorème de Kleene

Comme évoqué plus haut, le théorème de Kleene [9] affirme que l'ensemble des langages réguliers sur un alphabet Σ est exactement l'ensemble des langages sur Σ reconnaissables par automate fini. De plus, il existe des algorithmes permettant de construire un automate à partir d'une expression régulière et inversement. L'algorithme de Thompson [18] permet ainsi d'aller de l'expression à l'automate, tout comme la construction de Glushkov. La méthode de départ ou l'algorithme de McNaughton et Yamada [13] permettent au contraire d'aller de l'automate à l'expression.

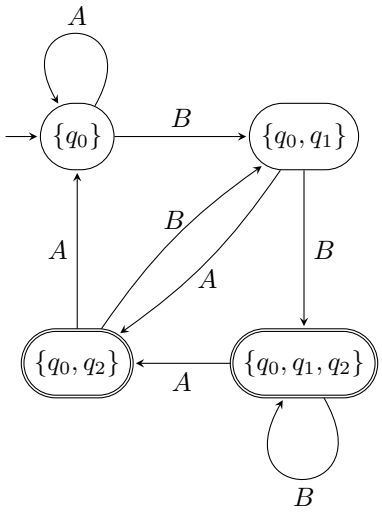

Figure 3.2. *Automate complet obtenu par la méthode* powerset construction *à partir de l'automate fini non-déterministe de la figure 3.1.*

Algorithme de Thompson

L'algorithme de Thompson consiste à construire un automate fini non-déterministe à partir d'une expression régulière, en utilisant des constructions pour l'union, la concaténation et l'étoile. Ces constructions font apparaître des transitions portant sur le mot vide ε, qui sont ensuite éliminées. À chaque expression régulière est associé un automate fini. Cet automate est construit par induction sur la structure de l'expression.

Exemple 3.4. Considérons l'expression régulière $\mathsf{E} = ab + c^*$ sur l'alphabet $\Sigma = \{a, b, c\}$.

On construit d'abord trois automates reconnaissant respectivement les langages $\{a\}$, $\{b\}$, $\{c\}$ (figure 3.3).

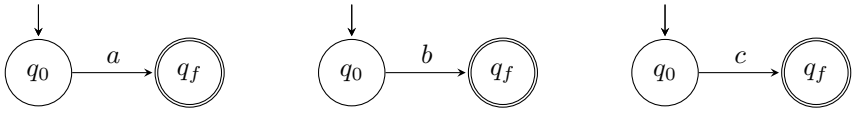

Figure 3.3. *Automates finis non-déterministes reconnaissant les langages $\{a\}$, $\{b\}$, $\{c\}$.*

Par concaténation des deux premiers automates, obtient un automate reconnaissant le langage $\{ab\}$ (figure 3.4).

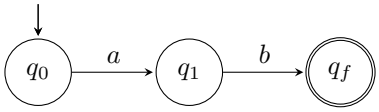

Figure 3.4. *Automate fini non-déterministe reconnaissant le langage $\{ab\}$.*

À partir de l'automate reconnaissant le langage $\{c\}$, on obtient ensuite un automate reconnaissant le langage c^* (figure 3.5).

Le langage $\mathcal{L}(\mathsf{E})$ est alors reconnu par l'automate union des deux étapes précédentes (figure 3.6).

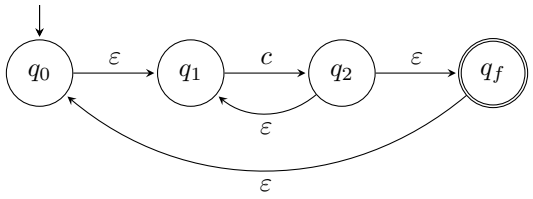

Figure 3.5. *Automate fini non-déterministe reconnaissant le langage c^*.*

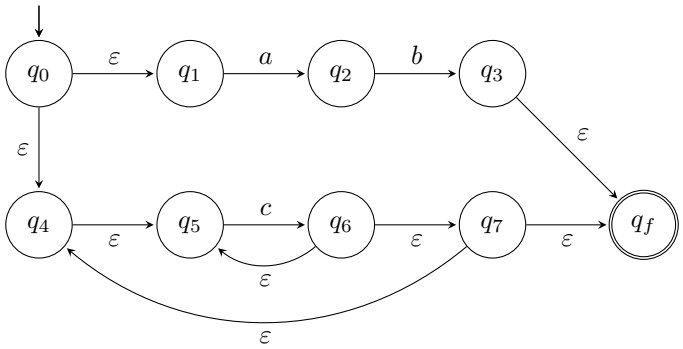

Figure 3.6. *Automate fini non-déterministe reconnaissant le langage $\mathcal{L}(ab + c^*)$.*

Pour finir, on supprime certaines transitions portant sur le mot vide ε. On obtient alors l'automate illustré sur la figure 3.7.

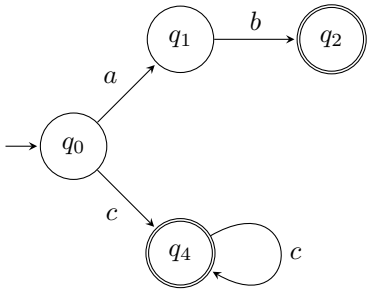

Figure 3.7. *Automate fini non-déterministe sans ε-transitions, reconnaissant le langage $\mathcal{L}(ab + c^*)$.*

•

Méthode de départ et Lemme de Arden

On peut déterminer le langage d'un automate complet par la méthode de départ et le Lemme de Arden. Pour simplifier, supposons ici que les états de l'automate $\mathcal{A}$ sont les entiers $1, 2, \ldots, n$ et que $\mathcal{A}$ admet l'état 1 comme unique état initial. Pour tout $k \in \{1, \ldots, n\}$, on note D_k l'ensemble des mots qui conduisent de l'état k à un état acceptant. Les ensembles D_k sont appelés *langages de départ*. Le langage $\mathcal{L}(\mathcal{A})$ coïncide alors avec l'ensemble D_1. Notons ensuite $\Sigma = \{A, B, \ldots\}$. Si k n'est pas un état acceptant, on pose l'équation

$$(E_k)\ D_k = \{A\}D_{\delta(k,A)} \cup \{B\}D_{\delta(k,B)} \cup \ldots.$$

Si au contraire k est un état acceptant, on pose l'équation

$$(E_k)\ D_k = \{\varepsilon\} \cup \{A\}D_{\delta(k,A)} \cup \{B\}D_{\delta(k,B)} \cup \dots.$$

On pose alors le système d'équations $(E_1), \dots, (E_n)$, qui exprime

$$\begin{cases} D_1 \text{ en fonction de } D_1, D_2, \dots, D_n, \\ D_2 \text{ en fonction de } D_1, D_2, \dots, D_n, \\ \vdots \\ D_n \text{ en fonction de } D_1, D_2, \dots, D_n. \end{cases}$$

Le Lemme de Arden permet de résoudre ce système.

Lemme 3.2 (Lemme de Arden)**.** Soient L_1 et L_2 deux langages. Le langage $L = L_1^* L_2$ est le plus petit langage (pour l'inclusion ensembliste) qui est solution de l'équation $X = (L_1 \cdot X) \cup L_2$. De plus, si L_1 ne contient pas le mot vide ε, alors le langage $L = L_1^* L_2$ est l'unique solution de cette équation.

Dans le système d'équations de la méthode de départ, certaines des équations permettent souvent d'effectuer des simplifications. Par exemple :

- si le second membre est le même dans la ligne $D_i = \dots$ et la ligne $D_j = \dots$, alors on peut en déduire la relation $D_i = D_j$;
- si l'une des équations est de la forme $D_i = \Sigma D_i$, on déduit du lemme de Arden que $D_i = \varnothing$ (en effet, si L est un langage, alors $L = L \cup \varnothing$ et $L\varnothing = \varnothing$ [2]). On dit alors que i est un état *piège*, ou *puits*).

Exemple 3.5. Considérons l'automate $\mathcal{A}$ sur $\Sigma = \{a, b\}$, illustré sur la figure 3.8. Le système

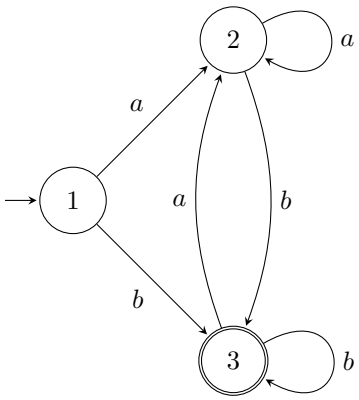

Figure 3.8. *Exemple d'automate fini dont on détermine le langage accepté par la méthode de départ.*

d'équations de la méthode de départ s'écrit :

$$\begin{cases} D_1 = \{a\}D_2 \cup \{b\}D_3, \\ D_2 = \{a\}D_2 \cup \{b\}D_3, \\ D_3 = \{\varepsilon\} \cup \{a\}D_2 \cup \{b\}D_3. \end{cases}$$

2. Le langage vide est absorbant pour la concaténation des langages (voir le cours *Langages et automates* de 3è année de Licence).

On a donc $D_3 = \{\varepsilon\} \cup D_2$, d'où $D_2 = \{a\}D_2 \cup \{b\}(\{\varepsilon\} \cup D_2)$, ce qui donne $D_2 = \{b\} \cup \{a,b\}D_2$. D'après le Lemme de Arden, on obtient $D_2 = \{a,b\}^*\{b\}$. Or, on a $\mathcal{L}(\mathcal{A}) = D_1$ et $D_1 = D_2$, d'où

$$\mathcal{L}(\mathcal{A}) = \{a,b\}^*\{b\}.$$

•

3.2 Vérification de propriétés de sûreté régulières

Dans cette section, nous allons montrer comment les automates finis peuvent être utilisés pour vérifier une classe importante de propriétés de sûreté. Ces propriétés de sûreté particulières sont caractérisées par le fait que leurs mauvais préfixes constituent un langage régulier, qui est donc reconnaissable par un automate fini non-déterministe. On les nomme donc naturellement *propriétés de sûreté régulières*. Le résultat principal de cette section est que la vérification d'une telle propriété de sûreté régulière sur un système de transition fini ST peut se ramener à une vérification d'invariant sur le produit de ST avec un automate fini non-déterministe $\mathcal{A}$ pour les mauvais préfixes. Autrement dit, si l'on souhaite vérifier une propriété de sûreté régulière pour un système de transition fini ST, il suffit de réaliser une analyse d'accessibilité sur le produit $ST \otimes \mathcal{A}$ pour vérifier un invariant associé sur $ST \otimes \mathcal{A}$.

3.2.1 Propriétés de sûreté régulières

Soit *Prop* un ensemble de propositions atomiques. Rappelons que les propriétés de sûreté sont des propriétés linéaires, c'est-à-dire des ensembles de mots infinis sur 2^{Prop}, telles que chaque trace qui ne satisfait pas une telle propriété de sûreté admet un mauvais préfixe qui cause une réfutation. Les mauvais préfixes sont finis, donc l'ensemble des mauvais préfixes constitue un langage de mots finis sur l'alphabet $\Sigma = 2^{Prop}$. Les entrées $A \in \Sigma$ de l'automate fini non-déterministe sont donc des ensembles de propositions atomiques (ce sont donc des sous-ensembles de *Prop*).

Par exemple, si $Prop = \{a,b\}$, alors on a

$$\Sigma = \{A_1, A_2, A_3, A_4\},$$

avec $A_1 = \{\,\}$, $A_2 = \{a\}$, $A_3 = \{b\}$ et $A_4 = \{a,b\}$ (on note $A_1 = \{\,\}$ pour désigner le sous-ensemble vide de *Prop* et ainsi le distinguer de l'expression régulière $\varnothing$ qui représente le langage vide).

Définition 3.6 (Propriété de sûreté régulière)**.** Soit *Prop* un ensemble de propositions atomiques et P_s une propriété de sûreté sur *Prop*. On dit que P_s est une propriété de sûreté *régulière* si son ensemble de mauvais préfixes constitue un langage régulier sur 2^{Prop}. □

Chaque invariant est une propriété de sûreté régulière. Si Φ est une formule propositionnelle d'un invariant, alors Φ doit être vérifiée par tous les états accessibles, et le langage des mauvais préfixes est constitué des mots finis $A_0 A_1 \dots A_n$ tels que $A_i \not\models \Phi$ pour un certain $i \in \{1, \dots, n\}$. Un tel langage est régulier, puisqu'il est caractérisé par l'expression régulière

$$\mathsf{E} = \tilde{\Phi}^*(\neg\tilde{\Phi})\mathsf{true}^*,$$

où $\tilde{\Phi}$ correspond à l'ensemble des parties A de *Prop* telles que $A \models \Phi$, $\neg\tilde{\Phi}$ à l'ensemble des parties A de *Prop* telles que $A \not\models \Phi$, et true représente l'ensemble de toutes les parties de *Prop*.

Le langage des mauvais préfixes d'une propriété invariante P_{inv} peut être représenté par un automate fini non-déterministe à deux états, tel que donné sur la figure 3.9. Dans cette figure, une arête du type

$$q \xrightarrow{\Psi} q'$$

représente toutes les transitions $q \xrightarrow{A} q'$ avec $A \models \Psi$.

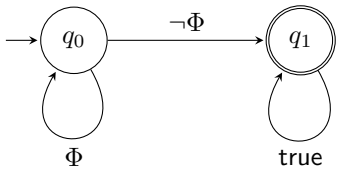

Figure 3.9. *Automate fini non-déterministe acceptant tous les mauvais préfixes de la propriété invariante associée à la formule* Φ.

L'ensemble des mauvais préfixes minimaux consitute également un langage régulier, caractérisé par l'expression régulière

$$\mathsf{F} = \tilde{\Phi}^*(\neg\tilde{\Phi}),$$

et reconnu par l'automate fini non-déterministe obtenu à partir de la figure 3.9 en supprimant la boucle true sur l'état q_1.

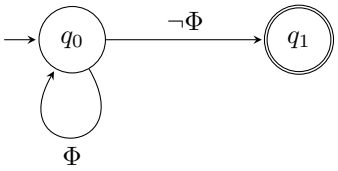

Figure 3.10. *Automate fini non-déterministe acceptant tous les mauvais préfixes minimaux de la propriété invariante associée à la formule* Φ.

Le lemme suivant montre que les propriétés de sûreté régulières peuvent être définies à partir des mauvais préfixes minimaux uniquement.

Lemme 3.3. Soit P_s une propriété de sûreté. Alors P_s est régulière si et seulement si l'ensemble de ses mauvais préfixes minimaux est régulier.

Démonstration. Supposons que l'ensemble M des mauvais préfixes minimaux de P_s est régulier. Soit alors $\mathcal{A} = \big(Q, 2^{Prop}, \delta, Q_0, F\big)$ un automate fini non-déterministe pour M. On construit un automate fini non-déterministe $\mathcal{A}'$ en ajoutant à $\mathcal{A}$ des boucles de type

$$q \xrightarrow{A} q,$$

pour tous les états $q \in F$ et pour toute partie A de l'ensemble *Prop*. On vérifie facilement que le langage accepté par $\mathcal{A}'$ est constitué de l'ensemble M' de tous les mauvais préfixes de P_s. Ainsi, M' est un langage régulier. Donc la propriété de sûreté P_s est bien régulière.

Supposons ensuite que P_s est une propriété de sûreté régulière. Soit alors $\mathcal{A}' = \big(Q, 2^{Prop}, \delta, Q_0, F\big)$ un automate fini déterministe pour l'ensemble M' des mauvais préfixes de P_s. Pour l'ensemble M des mauvais préfixes minimaux de P_s, on construit un automate fini déterministe $\mathcal{A}$ en supprimant toutes les transitions issues des états acceptants de $\mathcal{A}'$. Vérifions que $\mathcal{L}(\mathcal{A}) = M$.

Soit $w = A_1 \dots A_n \in \mathcal{L}(\mathcal{A})$; alors l'exécution $q_0 q_1 \dots q_n$ dans $\mathcal{A}$ acceptant w, est aussi une exécution acceptante de w dans $\mathcal{A}'$. Donc $w \in \mathcal{L}(\mathcal{A}')$. Donc w est un mauvais préfixe pour P_s. Supposons que w n'est pas un mauvais préfixe minimal. Alors il existe un préfixe $A_1 \dots A_i$ de w, avec $i < n$, qui est aussi un mauvais préfixe de P_s. Donc $A_1 \dots A_i \in \mathcal{L}(\mathcal{A}')$. Puisque $\mathcal{A}'$ est déterministe, l'unique exécution pour $A_1 \dots A_i$ dans $\mathcal{A}'$ est $q_0 \dots q_i$ et $q_i \in F$. Comme $i < n$ et q_i ne possède pas de transition sortante dans $\mathcal{A}$, alors $q_0 \dots q_i \dots q_n$ ne peut pas être une exécution pour $A_1 \dots A_i \dots A_n$ dans $\mathcal{A}$. Cela contredit l'hypothèse et montre que w est un mauvais préfixe minimal pour P_s.

Supposons enfin que w est un mauvais préfixe minimal pour P_s, et notons $q_0 \dots q_n$ l'unique exécution pour w dans $\mathcal{A}'$. On a $A_1 \dots A_n \in M' = \mathcal{L}(\mathcal{A}')$ et pour tout $i \in \{1, \dots, n-1\}$, $A_1 \dots A_i \notin M' = \mathcal{L}(\mathcal{A}')$. Donc $q_i \notin F$ pour $0 \leq i < n$, alors que $q_n \in F$. Par conséquent, $q_0 \dots q_n$ est une exécution acceptante pour w dans $\mathcal{A}$. Donc $w \in \mathcal{L}(\mathcal{A})$. □

Exemple 3.6. Soit $Prop = \{a, b\}$; considérons la formule $\Phi = a \vee \neg b$. Alors $\tilde{\Phi}$ correspond à l'expression régulière $\{\,\} + \{a\} + \{a, b\}$, $\neg\tilde{\Phi}$ correspond à l'expression régulière $\{b\}$, et true correspond à l'expression régulière $\{\,\} + \{a\} + \{b\} + \{a, b\}$. Ainsi, les mauvais préfixes de la propriété invariante correspondant à la formule $a \vee \neg b$ sont donnés par l'expression régulière

$$\mathsf{E} = \big(\{\,\} + \{a\} + \{a, b\}\big)^* \; \neg\{b\} \; \big(\{\,\} + \{a\} + \{b\} + \{a, b\}\big)^*.$$

Le langage $\mathcal{L}(\mathsf{E})$ est donc constitué des mots finis $A_0 A_1 \dots A_n$ tels que $A_i = \{b\}$ pour un certain $i \in \{1, \dots, n\}$.

L'automate fini non-déterministe donné sur la figure 3.9 est une représentation d'un automate à deux états q_0, q_1, avec pour transitions

$$\begin{aligned}
&q_0 \xrightarrow{\{\,\}} q_0, \quad q_0 \xrightarrow{\{a\}} q_0, \quad q_0 \xrightarrow{\{a,b\}} q_0,\\
&q_0 \xrightarrow{\{b\}} q_1,\\
&q_1 \xrightarrow{\{\,\}} q_1, \quad q_1 \xrightarrow{\{a\}} q_1, \quad q_1 \xrightarrow{\{b\}} q_1, \quad q_1 \xrightarrow{\{a,b\}} q_1.
\end{aligned}$$

Les mauvais préfixes minimaux sont décrits par l'expression régulière

$$\mathsf{F} = \big(\{\,\} + \{a\} + \{a, b\}\big)^* \; \neg\{b\}.$$

•

Exemple 3.7. Considérons à nouveau le modèle d'exclusion mutuelle par sémaphore de deux processus, et la propriété de sûreté P_s : « *il y a au plus un processus en état critique* ». Les mauvais préfixes de P_s forment le langage constitué de tous les mots finis $A_1 \dots A_n$ pour lesquels il existe (au moins) un indice $i \in \{1, \dots, n\}$ tel que

$$\{\mathsf{crit}_1, \mathsf{crit}_2\} \subseteq A_i.$$

Si n est le plus petit indice vérifiant cette propriété, c'est-à-dire

$$\{\mathsf{crit}_1, \mathsf{crit}_2\} \subseteq A_n \text{ et } \{\mathsf{crit}_1, \mathsf{crit}_2\} \not\subseteq A_j \text{ pour } 1 \leq j < n,$$

alors $A_1 \dots A_n$ est un mauvais préfixe minimal. Le langage de tous les mauvais préfixes minimaux est régulier et peut être reconnu par l'automate fini non-déterministe donné sur la figure 3.11. La propriété d'exclusion mutuelle P_s est donc bien une propriété de sûreté régulière. •

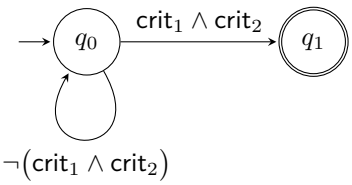

Figure 3.11. *Automate fini non-déterministe acceptant tous les mauvais préfixes minimaux de la propriété d'exclusion mutuelle.*

Exemple 3.8. Considérons à nouveau un feu de circulation avec trois couleurs possibles : rouge, orange, vert. Soit la propriété P_s : « *une phase rouge est toujours précédée immédiatement par une phase orange* ». Cette propriété correspond à l'ensemble des mots infinis $\sigma = A_0 A_1 \dots$ avec $A_i \subseteq \{\text{rouge}, \text{orange}\}$, tels que pour tout $i \geq 0$, on ait

$$\text{rouge} \in A_i \text{ implique } i > 0 \text{ et } \text{orange} \in A_{i-1}$$

(si le feu est dans une phase verte à l'indice i, alors $A_i = \{\,\}$). Les mauvais préfixes de P_s sont les mots finis qui ne respectent pas cette condition. Par exemple, le mot $\{\,\}\{\,\}\{\text{rouge}\}$ est un mauvais préfixe. Plus généralement, les mauvais préfixes minimaux de P_s sont les mots finis de la forme $A_1 \dots A_n$ tels que $n > 1$, $\text{rouge} \in A_n$ et $\text{orange} \notin A_{n-1}$. Ces mauvais préfixes minimaux sont reconnus par l'automate fini non-déterministe donné sur la figure 3.12.

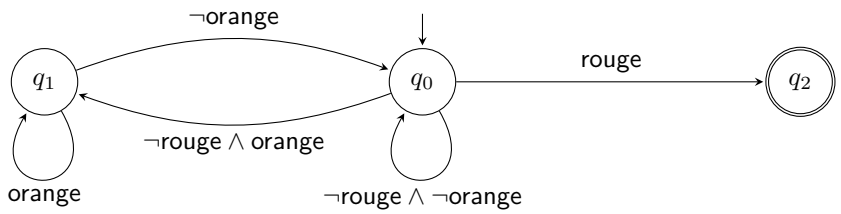

Figure 3.12. *Automate fini non-déterministe acceptant tous les mauvais préfixes minimaux d'une propriété d'un feu de circulation.*

Sur cette figure, les arêtes sont étiquetées sur l'alphabet $\Sigma = 2^{Prop}$ avec $Prop = \{\text{rouge}, \text{orange}\}$. On a donc

$$\Sigma = \big\{\{\,\}, \{\text{rouge}\}, \{\text{orange}\}, \{\text{rouge}, \text{orange}\}\big\}.$$

Par exemple, l'étiquette orange sur la boucle de l'état q_1 correspond à toutes les parties $A \subseteq Prop$ contenant orange, c'est-à-dire $\{\text{orange}\}$ ou $\{\text{orange}, \text{rouge}\}$ (certains types de feux peuvent être de deux couleurs en même temps). Autrement dit, la boucle de l'état q_1 résume deux transitions :

$$q_1 \xrightarrow{\{\text{orange}\}} q_1 \text{ et } q_1 \xrightarrow{\{\text{orange,rouge}\}} q_1.$$

De même, l'arête ¬orange de q_1 vers q_0 représente les transitions

$$q_1 \xrightarrow{\{\text{rouge}\}} q_0 \text{ et } q_1 \xrightarrow{\{\,\}} q_0.$$

Enfin, l'arête rouge de q_0 vers q_2 correspond aux étiquettes $\{\text{rouge}\}$ et $\{\text{rouge}, \text{orange}\}$; l'arête ¬rouge ∧ orange de q_0 vers q_1 correspond à l'étiquette $\{\text{orange}\}$; la boucle ¬rouge∧¬orange de l'état q_0 correspond à l'étiquette $\{\,\}$.

Ainsi, la propriété P_s est bien une propriété de sûreté régulière. •

Donnons enfin un exemple de propriété de sûreté qui n'est pas régulière.

Exemple 3.9. Considérons à nouveau un distributeur de boissons. Soit la propriété de sûreté P_s : « *le nombre de pièces insérées est toujours au moins égal au nombre de boissons distribuées* ». Posons $Prop = \{\text{paye}, \text{distribue}\}$. Les mauvais préfixes de P_s forment le langage

$$\Big\{\sigma \in \big(2^{Prop}\big)^{\omega} \mid occ(\sigma, \text{distribue}) > occ(\sigma, \text{paie})\Big\},$$

où $occ(\sigma, a)$ désigne le nombres d'occurences de a dans σ. On peut montrer que ce langage n'est pas régulier. •

3.2.2 Algorithme de vérification d'une propriété de sûreté régulière

Soit P_s une propriété de sûreté régulière sur un ensemble *Prop* de propositions atomiques, et soit $\mathcal{A}$ un automate fini non-déterministe reconnaissant les mauvais préfixes minimaux de P_s.

Si le mot vide ε appartient à $\mathcal{L}(\mathcal{A})$, alors tous les mots finis sur 2^{Prop} sont des mauvais préfixes, donc $P_s = \varnothing$. On suppose donc que $\varepsilon \notin \mathcal{L}(\mathcal{A})$.

Soit de plus ST un système de transition fini sans état terminal sur l'ensemble *Prop*. On souhaite établir une méthode algorithmique pour vérifier si ST satisfait la propriété P_s ou non.

D'après le Lemme 2.2, on sait que $ST \models P_s$ si et seulement si $\mathit{Traces}_{\mathit{fin}}(ST) \cap \mathit{MauvPre}(P_s) = \varnothing$, soit encore $\mathit{Traces}_{\mathit{fin}}(ST) \cap \mathcal{L}(\mathcal{A}) = \varnothing$. Il suffit donc de vérifier que $\mathit{Traces}_{\mathit{fin}}(ST) \cap \mathcal{L}(\mathcal{A}) = \varnothing$ pour établir que $ST \models P_s$.

Pour cela, on s'inspire de la méthode utilisée pour vérifier si deux langages de deux automates finis non-déterministes $\mathcal{A}_1$ et $\mathcal{A}_2$ ont une intersection vide. En effet, on a

$$\mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2) = \varnothing \Leftrightarrow \mathcal{L}(\mathcal{A}_1 \otimes \mathcal{A}_2) = \varnothing.$$

Or, déterminer si $\mathcal{L}(\mathcal{A}_1 \otimes \mathcal{A}_2) = \varnothing$ se réduit à une recherche d'accessibilité dans l'automate produit $\mathcal{A}_1 \otimes \mathcal{A}_2$.

Nous allons donc définir un produit de ST et $\mathcal{A}$, qui déterminera un système de transition noté $ST \otimes \mathcal{A}$. Pour ce nouveau système de transition $ST \otimes \mathcal{A}$, nous allons de plus construire une propriété invariante P_Φ, associée à une formule Φ obtenue à partir des états acceptants de $\mathcal{A}$, telle que la condition $\mathit{Traces}_{\mathit{fin}}(ST) \cap \mathcal{L}(\mathcal{A}) = \varnothing$ soit équivalente à $ST \otimes \mathcal{A} \models P_\Phi$. L'algorithme 2 pourra alors être utilisé pour vérifier si $ST \otimes \mathcal{A} \models P_\Phi$.

Posons donc $ST = (S, \mathit{Act}, \rightarrow, I, \mathit{Prop}, L)$ et $\mathcal{A} = (Q, 2^{Prop}, \delta, Q_0, F)$ avec $Q_0 \cap F = \varnothing$; l'alphabet de $\mathcal{A}$ est donc constitué d'ensembles de propositions atomiques. Le système de transitions $ST \otimes \mathcal{A}$ a pour ensemble d'états le produit $S' = S \times Q$, et sa relation de transition $\rightarrow'$ est telle que tout fragment de chemin initial $\pi = s_0 s_1 \dots s_n$ dans ST peut être prolongé en un fragment de chemin

$$< s_0, q_1 >< s_1, q_2 > \cdots < s_n, q_{n+1} >$$

dans $ST \otimes \mathcal{A}$, tel qu'il existe un état initial $q_0 \in Q_0$ pour lequel la suite

$$q_0 \xrightarrow{L(s_0)} q_1 \xrightarrow{L(s_1)} q_2 \xrightarrow{L(s_2)} \dots \xrightarrow{L(s_n)} q_{n+1}$$

est une exécution (non nécessairement acceptante) de $\mathcal{A}$ qui engendre le mot

$$\mathit{trace}(\pi) = L(s_0)L(s_1)\dots L(s_n).$$

Enfin, les étiquettes des états de $ST \otimes \mathcal{A}$ sont les noms des états de $\mathcal{A}$. On obtient la définition suivante.

Définition 3.7 (Produit d'un système de transition et d'un automate fini non-déterministe)**.** Soit $ST = (S, \mathit{Act}, \rightarrow, I, \mathit{Prop}, L)$ un système de transition fini sans état terminal sur un ensemble *Prop* de propositions atomiques, et soit $\mathcal{A} = (Q, 2^{Prop}, \delta, Q_0, F)$ un automate fini non-déterministe sur l'alphabet $\Sigma = 2^{Prop}$, tel que $Q_0 \cap F = \varnothing$. Le système de transition produit $ST \otimes \mathcal{A}$ est défini par

$$ST \otimes \mathcal{A} = (S', \mathit{Act}, \rightarrow', I', \mathit{Prop}', L'),$$

avec :

- $S' = S \times Q$,
- $\rightarrow'$ est la plus petite relation de transition définie par la règle

$$\frac{s \xrightarrow{\alpha} t \ \wedge \ q \xrightarrow{L(t)} p}{< s, q > \xrightarrow{\alpha}{}' < t, p >},$$

- $I' = \big\{ < s_0, q > \in S' \ ; \ s_0 \in I \ \wedge \ (\exists q_0 \in Q_0) \text{ tel que } q_0 \xrightarrow{L(s_0)} q \big\}$,

- $Prop' = Q$,
- $L'(< s, q >) = \{q\}$ pour tout $< s, q > \in S'$.

□

Pour définir correctement les propriétés linéaires (en particulier les propriétés invariantes), on a considéré des systèmes de transition sans état terminal. Or, même si ST ne possède pas d'état terminal, il se peut que $ST \otimes \mathcal{A}$ en possède. Cela peut se produire s'il existe dans $\mathcal{A}$ un état q qui n'a pas de successeur direct pour un ensemble A de propositions atomiques, soit $\delta(q, A) = \varnothing$. On peut éviter cette situation en supposant que $\delta(q, A) \neq \varnothing$ pour tout état $q \in Q$ et pour toute partie $A \subseteq Prop$. Cette hypothèse ne représente pas une restriction, car tout automate fini non-déterministe peut être transformé en un automate équivalent qui satisfait cette condition, en ajoutant un état q_{trap} et des transitions $q \xrightarrow{A} q_{trap}$ à chaque fois que l'on a $\delta(q, A) = \varnothing$. On peut aussi étendre la notion d'invariant aux systèmes de transition possédant des états terminaux.

Exemple 3.10. Considérons à nouveau un feu de circulation avec trois couleurs possibles : rouge, orange, vert. La propriété « *chaque phase rouge est immédiatement précédée d'une phase orange* » est une propriété de sûreté régulière, dont l'ensemble des mauvais préfixes minimaux est accepté par l'automate fini non-déterministe $\mathcal{A}$ donné sur la figure 3.12.

Supposons que ce feu de circulation soit de type Allemand, avec la possibilité d'indiquer les couleurs rouge et orange simultanément, afin de signifier que le feu va bientôt passer au vert.

Soit alors $Prop = \{\text{rouge}, \text{orange}\}$. On définit une fonction d'étiquetage L en posant

$$L(\text{rouge}) = \{\text{rouge}\}, L(\text{orange}) = \{\text{orange}\}, L(\text{vert}) = L(\text{rouge/orange}) = \varnothing.$$

Le système de transition ST correspondant à un tel feu de circulation est indiqué sur la figure 3.13, ainsi que le système de transition $ST \otimes \mathcal{A}$, où $\mathcal{A}$ est donné sur la figure 3.12. On doit noter que l'état rouge/orange est bien compatible avec la transition

$$q_0 \xrightarrow{\neg\text{rouge} \wedge \neg\text{orange}} q_0$$

de $\mathcal{A}$. Le produit $ST \otimes \mathcal{A}$ admet 4 états accessibles. •

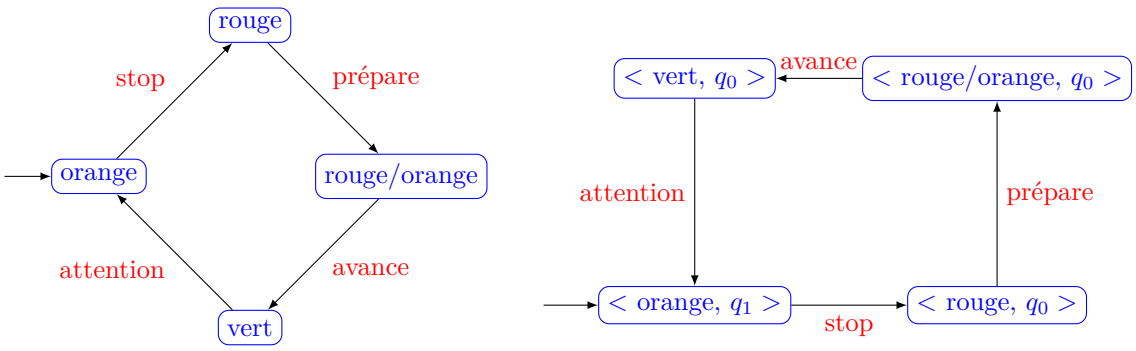

Figure 3.13. *Système de transition ST modélisant le fonctionnement d'un feu de circulation de type Allemand (à gauche), et système de transition produit $ST \otimes \mathcal{A}$ (à droite).*

Le théorème suivant montre que la vérification d'une propriété de sûreté régulière peut se ramener à la vérification d'une propriété invariante dans le système de transition produit.

Théorème 3.2. Soit $ST = (S, Act, \rightarrow, I, Prop, L)$ un système de transition fini sans état terminal sur un ensemble $Prop$ de propositions atomiques, et soit $\mathcal{A} = (Q, 2^{Prop}, \delta, Q_0, F)$ un automate fini non-déterministe sur l'alphabet $\Sigma = 2^{Prop}$, tel que $Q_0 \cap F = \varnothing$. Soit P_s une propriété de sûreté régulière sur $Prop$, telle que $\mathcal{L}(\mathcal{A})$ soit constitué des mauvais préfixes (minimaux) de P_s. Alors les assertions suivantes sont équivalentes :

- $ST \models P_s$,
- $Traces_{fin}(ST) \cap \mathcal{L}(\mathcal{A}) = \varnothing$,
- $ST \otimes \mathcal{A} \models P_{inv(\mathcal{A})}$,

où $P_{inv(\mathcal{A})}$ est la propriété invariante associée à la formule

$$\bigwedge_{q \in F} \neg q.$$

Démonstration. Notons $\neg F = \bigwedge_{q \in F} \neg q$; $\neg F$ correspond donc aux états non acceptants de $\mathcal{A}$. Comme nous l'avons remarqué au début de cette section, on a déjà l'équivalence entre (a) et (b) d'après le Lemme 2.2. Il suffit donc de montrer que (c) implique (a) et que (b) implique (c).

Montrons (par contraposée) que (c) implique (a). Si $ST \not\models P_s$, alors il existe un fragment de chemin initial fini $\hat{\pi} = s_0 s_1 \dots s_n$ dans ST tel que

$$trace(\hat{\pi}) = L(s_0)L(s_1)\dots L(s_n) \in \mathcal{L}(\mathcal{A}).$$

Donc il existe dans $\mathcal{A}$ une exécution acceptante $q_0 q_1 \dots q_{n+1}$ pour $trace(\hat{\pi})$, avec $q_0 \in Q_0$, $q_{n+1} \in F$ et $q_i \xrightarrow{L(s_i)} q_{i+1}$ pour $0 \leq i \leq n$. Mais alors, la suite

$$< s_0, q_1 >< s_1, q_2 > \cdots < s_n, q_{n+1} >$$

est un fragment de chemin initial dans $ST \otimes \mathcal{A}$ tel que $< s_n, q_{n+1} > \not\models \neg F$. Donc $ST \otimes \mathcal{A} \not\models P_{inv(\mathcal{A})}$.

Montrons enfin (à nouveau par contraposée) que (c) implique (a). Supposons pour cela que

$$ST \otimes \mathcal{A} \not\models P_{inv(\mathcal{A})}.$$

Alors il existe dans $ST \otimes \mathcal{A}$ un fragment de chemin initial

$$< s_0, q_1 >< s_1, q_2 > \cdots < s_n, q_{n+1} >$$

avec $q_{n+1} \in F$. Ainsi, $s_0 s_1 \dots s_n$ est un fragment de chemin initial dans ST, avec $q_i \xrightarrow{L(s_i)} q_{i+1}$ pour $1 \leq i \leq n$. Or, $< s_0, q_1 >$ est un état initial de $ST \otimes \mathcal{A}$, donc il existe $q_0 \in Q_0$ tel que

$$q_0 \xrightarrow{L(s_0)} q_1.$$

Donc $trace(s_0 s_1 \dots s_n) \in Traces_{fin}(ST) \cap \mathcal{L}(\mathcal{A})$, ce qui prouve

$$Traces_{fin}(ST) \cap \mathcal{L}(\mathcal{A}) \neq \varnothing.$$

□

Le théorème 3.2 montre que pour vérifier si ST satisfait bien la propriété de sûreté régulière P_s, il suffit de vérifier qu'il n'existe pas d'état accessible $< s, q >$ dans le produit $ST \otimes \mathcal{A}$, où q serait un état acceptant de $\mathcal{A}$. La propriété invariante « *ne jamais visiter un état acceptant dans* $\mathcal{A}$ » est donnée par la condition invariante $\Phi = \neg F$ et peut être vérifiée en exécutant l'algorithme 2 de vérification d'invariant par recherche en profondeur. De plus, si la propriété de sûreté régulière P_s est réfutée, alors un contre-exemple est obtenu, sous la forme d'un fragment de chemin fini $< s_0, q_1 >< s_1, q_2 > \cdots < s_n, q_{n+1} >$ dans $ST \otimes \mathcal{A}$ menant à un état acceptant de $\mathcal{A}$, et qui à son tour détermine un fragment de chemin initial fini $s_0 s_1 \dots s_n$ dans ST, dont la trace est acceptée par $\mathcal{A}$. Cette trace est finalement un mauvais préfixe de la propriété P_s.

On obtient le corollaire suivant.

Algorithme 3 Vérification d'une propriété de sûreté régulière

Entrée : système de transition fini ST et propriété de sûreté régulière P_s.
Sortie : « OUI » si ST satisfait P_s, autrement « NON » et un contre-exemple.

Déterminer un automate fini non-déterministe $\mathcal{A}$ tel que $\mathcal{L}(\mathcal{A})$ corresponde aux mauvais préfixes (minimaux) de P_s
Construire le système de transition $ST \otimes \mathcal{A}$
Vérifier sur $ST \otimes \mathcal{A}$ l'invariant $P_{inv(\mathcal{A})}$ avec la proposition $\neg F = \bigwedge_{q \in F} \neg q$

Si $ST \otimes \mathcal{A} \models P_{inv(\mathcal{A})}$ **alors**
  renvoyer VRAI
Sinon
  Déterminer un fragment de chemin initial $< s_0, q_1 > \cdots < s_n, q_{n+1} >$ dans $ST \otimes \mathcal{A}$ avec $q_{n+1} \in F$
  renvoyer (FAUX, $s_0 s_1 \dots s_n$)
Fin Si

Corollaire 3.1. Soit $ST = (S, Act, \rightarrow, I, Prop, L)$ un système de transition fini sans état terminal sur un ensemble $Prop$ de propositions atomiques, et soit $\mathcal{A} = (Q, 2^{Prop}, \delta, Q_0, F)$ un automate fini non-déterministe sur l'alphabet $\Sigma = 2^{Prop}$, tel que $Q_0 \cap F = \varnothing$. Soit P_s une propriété de sûreté régulière sur $Prop$, telle que $\mathcal{L}(\mathcal{A})$ soit constitué des mauvais préfixes (minimaux) de P_s. Alors, pour tout fragment de chemin initial $< s_0, q_1 >< s_1, q_2 > \cdots < s_n, q_{n+1} >$ dans $ST \otimes \mathcal{A}$:

$$q_1, \dots, q_n \notin F \text{ et } q_{n+1} \in F \text{ implique } trace(s_0 s_1 \dots s_n) \in \mathcal{L}(\mathcal{A}).$$

L'algorithme 3 montre comment mettre en œuvre cette méthode.

Exemple 3.11. Considérons à nouveau un feu de circulation de type Allemand et la propriété de sûreté régulière P_s spécifiant que chaque phase rouge doit être immédiatement précédée d'une phase orange. Le système de transition $ST \otimes \mathcal{A}$ est donné sur la figure 3.13. On observe qu'aucun état dans $ST \otimes \mathcal{A}$ de la forme $< \dots, q_2 >$ n'est accessible. Donc l'invariant $\neg q_2$ est satisfait dans tous les états de $ST \otimes \mathcal{A}$. Par conséquent, le feu de circulation satisfait bien P_s.

Si le système de transition modélisant le feu de circulation est modifié, de sorte que l'état rouge est l'état initial, alors la propriété P_s n'est plus satisfaite. En effet, dans ce cas, l'invariant $\neg q_2$ n'est pas satisfait dans l'état initial. •

Pour conclure cette partie, nous donnons une borne de la complexité en temps et en espace de l'algorithme de vérification de propriétés de sûreté régulières.

Théorème 3.3. La complexité en temps et en espace de l'algorithme 3 est d'ordre $\mathcal{O}\big(|ST| \times |\mathcal{A}|\big)$, où $|ST|$ et $|\mathcal{A}|$ donnent le nombre d'états et de transitions dans ST et $\mathcal{A}$ respectivement.

Démonstration. La complexité en temps et en espace de l'algorithme 2 de vérification d'invariant sur $ST \otimes \mathcal{A}$ est d'ordre $\mathcal{O}(|ST \otimes \mathcal{A}|)$. De plus, le nombre $|ST \otimes \mathcal{A}|$ d'états et de transitions dans $ST \otimes \mathcal{A}$ est d'ordre $\mathcal{O}(|ST| \times |\mathcal{A}|\,)$. D'où l'ordre de complexité annoncé. □

3.3 Vérification de propriétés de sûreté non régulières

L'algorithme 3 de vérification de propriétés de sûreté régulières repose sur le fait que les automates finis acceptent des mots finis. On peut néanmoins étendre la méthode à des propriétés linéaires plus générales, qui décrivent de façon pertinente le comportement de systèmes réels. Certaines de ces propriétés peuvent être décrites avec des expressions dites *ω-régulières*. Pour cela, on considère une

variante des automates finis non-déterministes, appelés *automates de Büchi*. La syntaxe d'un automate de Büchi est identique à celle d'un automate fini non-déterministe. En revanche, le langage accepté d'un automate de Büchi, noté $\mathcal{L}_\omega(\mathcal{B})$, est infini ; le critère d'acceptance pour un tel automate est que l'ensemble des états acceptants doit être visité une infinité de fois.

Un automate de Büchi $\mathcal{B}$ reconnaît alors les « mauvaises traces » d'une propriété P à vérifier ; une analyse de graphe dans le produit de $\mathcal{B}$ avec le système de transition considéré, sur une propriété associée, dite de *persistance*, suffit alors pour vérifier si la propriété P est satisfaite ou non.

3.4 Exercices

Exercice 1

Sur l'alphabet $\Sigma = \{a, b\}$, on définit les langages $\mathcal{L} = \{a, ab\}$ et $\mathcal{L}' = \{a, b, ba\}$.

1. Déterminer les langages $\mathcal{L} \cup \mathcal{L}'$, $\mathcal{L}\mathcal{L}'$, $\mathcal{L}'\mathcal{L}$, $\mathcal{L}\mathcal{L}'\mathcal{L}$, $\mathcal{L}^3$.
2. Montrer qu'un mot de $\mathcal{L}^*$ ne contient jamais deux lettres b qui se suivent.
3. Soit $\mathcal{L}''$ le langage sur l'alphabet Σ formé des mots qui ne contiennent jamais deux lettres b qui se suivent. Trouver des mots de $\mathcal{L}''$ qui n'appartiennent pas à $\mathcal{L}^*$.
4. Le langage $\mathcal{L}''$ est-il régulier ?

Exercice 2

On considère l'automate $\mathcal{A}$ défini par $Q = \{1, 2, 3, 4\}$, $\Sigma = \{a, b\}$, $Q_0 = \{1\}$, $F = \{4\}$ et dont la fonction de transition δ est donnée par le tableau suivant.

	a	b
1	2	3
2	4	4
3	3	3
4	3	3

Représenter cet automate et déterminer son langage.

Exercice 3

Soit $Prop = \{a, b, c\}$. On considère le système de transition ST sur $Prop$ et l'automate fini non-déterministe $\mathcal{A}$ sur 2^{Prop}, représentés sur la figure 3.14 ci-dessous. Construire leur produit $ST \otimes \mathcal{A}$.

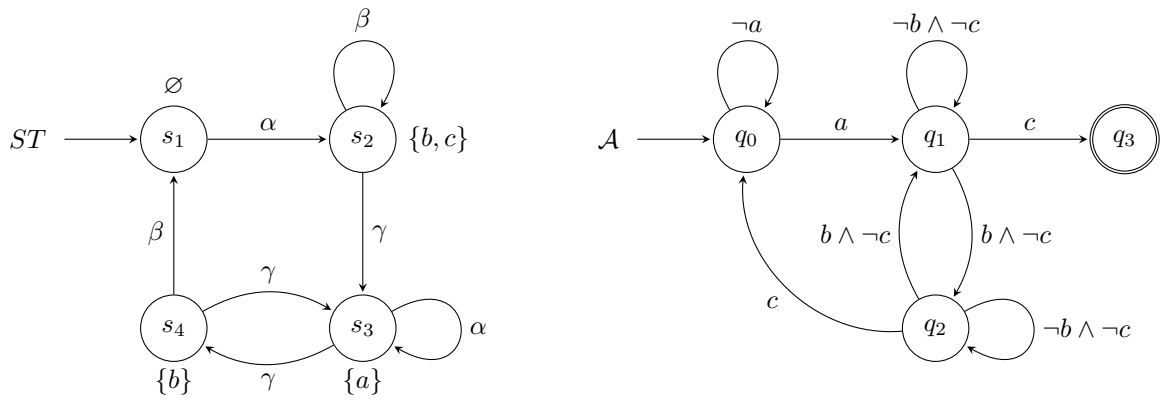

Figure 3.14. *Un système de transition ST sur Prop* $= \{a, b, c\}$ *et un automate fini non-déterministe* $\mathcal{A}$.

Exercice 4

Soit *Prop* $= \{\mathsf{rouge}, \mathsf{vert}\}$. On considère le système de transition ST sur *Prop* et l'automate fini non-déterministe $\mathcal{A}$ sur 2^{Prop}, représentés sur la figure 3.15 ci-dessous. Construire leur produit $ST \otimes \mathcal{A}$.

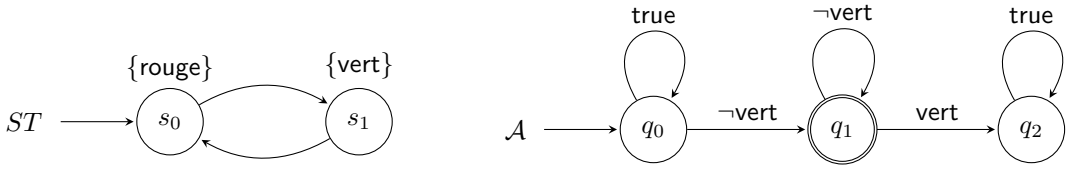

Figure 3.15. *Un système de transition ST sur Prop* $= \{\mathsf{rouge}, \mathsf{vert}\}$ *et un automate fini non-déterministe* $\mathcal{A}$.

Exercice 5

On considère le système de transition ST_{sem} modélisant l'exclusion mutuelle par sémaphore binaire de deux processus P_1, P_2, et le système de transition ST_{Peter} modélisant l'algorithme d'exclusion de Peterson.

1. Soit *Prop* $= \{\mathsf{pause}_1, \mathsf{crit}_1\}$ et Φ_1 la propriété « *le processus* P_1 *n'entre jamais en état critique directement depuis son état non-critique* ».
   (a) Pourquoi la propriété Φ_1 est-elle une propriété de sûreté régulière ?
   (b) Construire un automate fini non-déterministe pour les mauvais préfixes minimaux de Φ_1.
   (c) Le système de transition ST_{sem} vérifie-t-il la propriété Φ_1 ?
2. Soit *Prop*$' = \{\mathsf{crit}_1, x = 2\}$, et Φ_2 la propriété « *le processus* P_1 *n'entre jamais en état critique depuis un état où* $x = 2$ ».
   (a) Pourquoi la propriété Φ_2 est-elle aussi une propriété de sûreté régulière ?
   (b) Construire un automate fini non-déterministe pour les mauvais préfixes minimaux de Φ_2.
   (c) Prouver que le système de transition ST_{Peter} ne vérifie pas la propriété Φ_2 et donner un contrexemple.

Exercice 6

On considère un écosystème biologique admettant trois états : un état de persistance noté P, un état d'extinction noté E et un état transitoire noté T. Soit $Prop = \{\mathsf{T}, \mathsf{E}\}$. En l'absence d'événement climatique extrême, l'écosystème présente une dynamique biologique modélisée par le système de transition ST_{nat} de la figure 3.16.

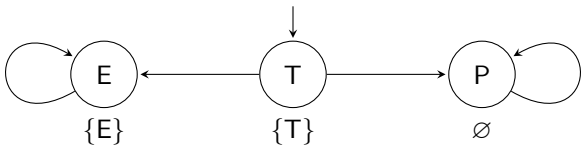

Figure 3.16. *Système de transition modélisant la dynamique biologique d'un écosystème.*

On considère la propriété Φ : « *si le système atteint l'état de persistance, alors il ne peut plus entrer en état d'extinction* ».

1. Que dire de la propriété Φ ?
2. Le système de transition ST_{nat} vérifie-t-il la propriété Φ ?
3. On suppose que l'écosystème est perturbé par des événements climatiques extrêmes, qui sont intégrés au modèle en ajoutant une transition possible de l'état P vers l'état T. Le système de transition obtenu vérifie-t-il la propriété Φ ?

CHAPITRE 4

Logique Temporelle Linéaire

Ce chapitre est une introduction à la Logique Temporelle Linéaire (LTL, *Linear Temporal Logic* en anglais), un formalisme logique adapté à la spécification des propriétés linéaires. La syntaxe et la sémantique de ce système logique sont présentées et illustrées par des exemples repris des chapitres précédents, afin de montrer son niveau d'expressivité. La deuxième partie du chapitre est consacrée à la Logique Temporelle Linéaire Bornée (BLTL, *Bounded Linear Temporal Logic* en anglais), une variante de la logique LTL adaptée à la spécification de propriétés de systèmes temporisés dont le comportement est observé sur un ensemble fini d'instants.

4.1 Syntaxe, sémantique et propriétés de la logique LTL

La logique LTL est une extension de la logique propositionnelle, proposée par Amir Pnueli en 1977 pour la vérification de programmes informatiques [14]. Cette logique permet plus généralement d'exprimer des propriétés de systèmes dont le comportement est réactif au cours du temps. Le terme *temporel* suggère donc un lien entre le comportement du système et le déroulement du temps. Ce lien permet d'intégrer l'ordre chronologique des événements observés, mais ne tient pas compte de leurs aspects quantitatifs, notamment de leurs durées.

4.1.1 Syntaxe LTL

Les ingrédients de base permettant de construire les formules de la logique LTL sont :

- un ensemble de propositions atomiques *Prop*,
- les connecteurs booléens de conjonction $\wedge$ et de négation $\neg$,
- deux opérateurs temporels notés $\bigcirc$ ("*next*") et $\mathcal{U}$ ("*until*").

L'opérateur $\bigcirc$ est unaire ; si φ est une formule LTL, alors $\bigcirc\varphi$ est vraie à l'instant présent lorsque φ est vraie à l'instant suivant. L'opérateur $\mathcal{U}$ est binaire ; si φ_1, φ_2 sont deux formules LTL, alors $\varphi_1 \mathcal{U} \varphi_2$ est vraie à l'instant présent lorsqu'il existe un instant futur pour lequel φ_2 sera vraie, et que φ_1 est vraie (au moins) jusqu'à cet instant futur.

Définition 4.1 (Syntaxe LTL)**.** Soit *Prop* un ensemble de propositions atomiques. Les formules LTL sur *Prop* sont construites à partir des quatre règles suivantes :

- true est une formule LTL,
- toute proposition atomique $a \in Prop$ est une formule LTL,
- si φ est une formule LTL, alors $\neg\varphi$ et $\bigcirc\varphi$ sont aussi des formules LTL,

- si φ_1, φ_2 sont des formules LTL, alors $\varphi_1 \wedge \varphi_2$ et $\varphi_1 \mathcal{U} \varphi_2$ sont aussi des formules LTL.

□

On convient que les opérateurs unaires sont prioritaires sur les opérateurs binaires. Ainsi, on a par exemple $\neg\varphi_1 \mathcal{U} \bigcirc \varphi_2 = (\neg\varphi_1)\mathcal{U}(\bigcirc\varphi_2)$. Les opérateurs binaires de disjonction $\vee$, d'implication $\rightarrow$, d'équivalence $\leftrightarrow$ et de parité $\oplus$ (ou exclusif) sont définis par

$$\begin{aligned}
\varphi_1 \vee \varphi_2 &= \neg(\neg\varphi_1 \wedge \neg\varphi_2),\\
\varphi_1 \rightarrow \varphi_2 &= \neg\varphi_1 \vee \varphi_2,\\
\varphi_1 \leftrightarrow \varphi_2 &= (\varphi_1 \rightarrow \varphi_2) \wedge (\varphi_2 \rightarrow \varphi_1),\\
\varphi_1 \oplus \varphi_1 &= (\varphi_1 \wedge \neg\varphi_2) \vee (\varphi_2 \wedge \neg\varphi_1),
\end{aligned}$$

pour toutes formules LTL φ_1, φ_2.

On définit également deux opérateurs temporels $\Diamond$ et $\Box$, pour toute formule LTL φ, par

$$\Diamond\varphi = \mathsf{true}\,\mathcal{U}\varphi, \qquad \Box\varphi = \neg\Diamond\neg\varphi.$$

La formule $\Diamond\varphi$ signifie qu'il existe un moment futur où φ sera vraie. L'opérateur $\Diamond$ est lu *eventually* en anglais, ce qui ne doit pas être traduit par *éventuellement* en français (l'adverbe «*éventuellement*» n'implique aucune obligation future, contrairement à "*eventually*"). On peut lire $\Diamond$ en prononçant « *à un moment, il y aura* ». La formule $\Box\varphi$ signifie que φ sera *toujours* vraie dans le futur.

La figure 4.1 illustre quelques éléments constitutifs essentiels de la logique LTL.

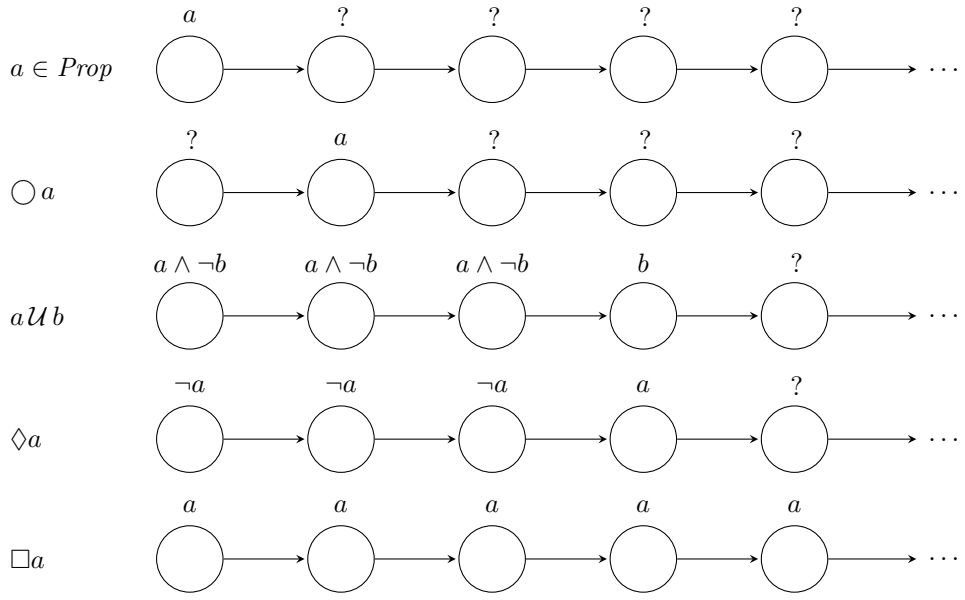

Figure 4.1. *Illustration de quelques éléments constitutifs essentiels de la logique LTL.*

On peut également associer les opérateurs $\Diamond$ et $\Box$. Ainsi, la formule $\Box\Diamond\varphi$ signifie que φ sera vraie infiniment souvent, et la formule $\Diamond\Box\varphi$ signifie que φ sera toujours vraie à partir d'un certain moment futur.

Exemple 4.1. Considérons la propriété d'exclusion mutuelle de deux processus P_1, P_2 admettant des états d'attente pause_1, pause_2 et des états critiques crit_1, crit_2 respectivement. La propriété de sûreté exprimant que P_1 et P_2 ne sont jamais simultanément en état critique correspond à la formule LTL

$$\Box(\neg\mathsf{crit}_1 \vee \neg\mathsf{crit}_2).$$

Puis, la propriété de vivacité exprimant que chaque processus est infiniment souvent en état critique correspond à la formule LTL

$$(\Box\Diamond\mathsf{crit}_1) \wedge (\Box\Diamond\mathsf{crit}_2).$$

La formule LTL suivante exprime le fait que chaque processus en état d'attente entre effectivement en état critique dans le futur :

$$(\Box\Diamond\mathsf{pause}_1 \rightarrow \Box\Diamond\mathsf{crit}_1) \wedge (\Box\Diamond\mathsf{pause}_2 \rightarrow \Box\Diamond\mathsf{crit}_2).$$

Enfin, si le protocole d'exclusion implique un sémaphore binaire y, alors la formule LTL

$$\Box\big((y = 0) \rightarrow \mathsf{crit}_1 \vee \mathsf{crit}_2\big)$$

décrit le fait que si le sémaphore y admet la valeur 0, alors un des deux processus entre en état critique. •

Exemple 4.2. Considérons un feu de circulation classique. La formule LTL $\Box\Diamond\mathsf{vert}$ correspond à la propriété de vivacité ≪ *le feu est vert infiniment souvent* ≫. La condition ≪ *si le feu est vert, alors il ne peut pas devenir rouge immédiatement* ≫ peut s'exprimer par la formule LTL

$$\Box(\mathsf{vert} \rightarrow \neg \bigcirc \mathsf{rouge}).$$

•

Remarques. Il existe dans la littérature scientifique des notations différentes pour les opérateurs de la logique LTL. Ainsi, les opérateurs $\bigcirc$, $\Diamond$, $\Box$ sont parfois notés respectivement X (pour *neXt*), F (pour *Finally*), G (pour *Globally*).

On peut aussi définir des opérateurs temporels $\bigcirc^{-1}$, $\Diamond^{-1}$ et $\Box^{-1}$, opérateurs inverses des opérateurs temporels $\bigcirc$, $\Diamond$ et $\Box$ respectivement, pour décrire des événements du passé. ◁

4.1.2 Sémantique LTL

Les formules LTL sont des formules qui décrivent des traces. Pour déterminer précisément si une trace satisfait une propriété LTL, on commence par définir une interprétation pour les *mots* infinis sur l'alphabet 2^{Prop} ; on définit ensuite une interprétation sur les *chemins* et les *états* d'un système de transition.

Définition 4.2 (Sémantique LTL sur les mots)**.** Soit φ une formule LTL sur un ensemble *Prop* de propositions atomiques. La propriété linéaire induite par φ est définie par

$$\mathit{Mots}(\varphi) = \big\{\sigma \in \big(2^{Prop}\big)^{\omega} \mid \sigma \models \varphi\big\},$$

où la relation de satisfaction $\models$ est définie pour tout mot $\sigma = A_0A_1A_2\ldots$ par les règles suivantes :

$$\begin{array}{lcl}
\sigma \models \mathsf{true}, & & \\
\sigma \models a & \Leftrightarrow & A_0 \models a \quad (\Leftrightarrow a \in A_0), \\
\sigma \models \varphi_1 \wedge \varphi_2 & \Leftrightarrow & \sigma \models \varphi_1 \text{ et } \sigma \models \varphi_2, \\
\sigma \models \neg\varphi & \Leftrightarrow & \sigma \not\models \varphi, \\
\sigma \models \bigcirc\varphi & \Leftrightarrow & A_1A_2A_3\ldots \models \varphi, \\
\sigma \models \varphi_1 \,\mathcal{U}\, \varphi_2 & \Leftrightarrow & \exists j \geq 0, A_jA_{j+1}A_{j+2}\ldots \models \varphi_2 \text{ et } A_iA_{i+1}A_{i+2}\ldots \models \varphi_1, 0 \leq i < j,
\end{array}$$

avec $a \in \mathit{Prop}$ et φ, φ_1, φ_2 des formules LTL. □

Remarque. Dans la règle définissant la sémantique de $\varphi_1\mathcal{U}\varphi_2$, on ne peut pas remplacer le suffixe infini $A_jA_{j+1}A_{j+2}\ldots$ par A_j. ◁

Pour les opérateurs temporels $\Diamond$ et $\Box$, on obtient

$$\begin{array}{lcl} \sigma \models \Diamond\varphi, & \Leftrightarrow & \exists j \geq 0, A_j A_{j+1} A_{j+2} \ldots \models \varphi \\ \sigma \models \Box\varphi & \Leftrightarrow & \forall j \geq 0, A_j A_{j+1} A_{j+2} \ldots \models \varphi. \end{array}$$

Après avoir défini la sémantique LTL sur les mots, on définit la sémantique LTL pour un système de transition. La définition suivante s'appuie sur la définition 2.7 de la relation de satisfaction pour une propriété linéaire. Rappelons que si s est un état d'un système de transition ST, alors $Paths(s)$ désigne l'ensemble des fragments de chemins maximaux qui commencent en s.

Définition 4.3 (Sémantique LTL sur les chemins et les états)**.** Soit $ST = (S, Act, \rightarrow, I, Prop, L)$ un système de transition sans état final et soit φ une formule LTL sur $Prop$.

Si π est un fragment de chemin infini de ST, on pose :

$$\pi \models \varphi \Leftrightarrow trace(\pi) \models \varphi.$$

Si $s \in S$, on pose :

$$s \models \varphi \Leftrightarrow \forall \pi \in Paths(s), \pi \models \varphi.$$

Enfin, on dit que ST satisfait φ, et on note $ST \models \varphi$, si

$$Traces(ST) \subseteq Mots(\varphi).$$

□

D'après la définition 2.7, on a

$$Traces(ST) \subseteq Mots(\varphi) \Leftrightarrow ST \models Mots(\varphi),$$

ce qui, d'après la définition de la propriété linéaire $Mots(\varphi)$, équivaut à

$$\pi \models \varphi, \forall \pi \in Paths(ST).$$

Enfin, d'après la définition de la relation de satisfaction $\models$ pour les états, on obtient :

$$ST \models \varphi \Leftrightarrow s_0 \models \varphi, \forall s_0 \in I.$$

Exemple 4.3. Considérons le système de transition ST sur $Prop = \{a, b\}$, représenté sur la figure 4.2.

On a de façon évidente $s_1 \models \Box a$, $s_2 \models \Box a$ et $s_3 \models \Box a$, donc $ST \models \Box a$.

Puis, on a $s_1 \models \bigcirc(a \wedge b)$, car $s_2 \models a \wedge b$ et s_2 est le seul successeur de s_1 ; cependant, on a $s_2 \not\models \bigcirc(a \wedge b)$ car s_3 est un successeur de s_2 tel que $s_3 \not\models a \wedge b$; de même, on a $s_3 \not\models \bigcirc(a \wedge b)$. Ainsi, s_3 est un état initial tel que $s_3 \not\models \bigcirc(a \wedge b)$. Donc $ST \not\models \bigcirc(a \wedge b)$.

De la même façon, on a $ST \models \Box\big(\neg b \rightarrow \Box(a \wedge \neg b)\big)$, puisque s_3 est le seul état initial tel que $s_3 \models \neg b$, s_3 est absorbant et s_3 vérifie $a \wedge \neg b$.

Enfin, on a $ST \not\models b\,\mathcal{U}\,(a \wedge \neg b)$, car le chemin initial $(s_1 s_2)^\omega$ ne visite aucun état pour lequel $a \wedge \neg b$ est vraie. On remarque néanmoins que tout chemin initial de la forme $(s_1 s_2)^* s_3^\omega$ satisfait la propriété $b\,\mathcal{U}\,(a \wedge \neg b)$.

•

Remarque. Pour un chemin π et une formule LTL, on a

$$\pi \models \varphi \Leftrightarrow \pi \not\models \neg\varphi.$$

Toutefois, on doit prendre garde que les assertions $ST \models \varphi$ et $ST \not\models \neg\varphi$ ne sont pas équivalentes en général. En effet, considérons le système de transition ST sur $Prop = \{a\}$, représenté sur la figure 4.3. Pour ce système de transition, on a $ST \not\models \Diamond a$, puisque le chemin initial $s_0(s_2)^\omega$ ne satisfait pas $\Diamond a$, et $ST \not\models \neg\Diamond a$, puisque le chemin initial $s_0(s_1)^\omega$ ne satisfait pas $\neg\Diamond a$.

Pour un système de transition ST, on peut seulement affirmer que $ST \models \neg\varphi$ implique $ST \not\models \varphi$. ◁

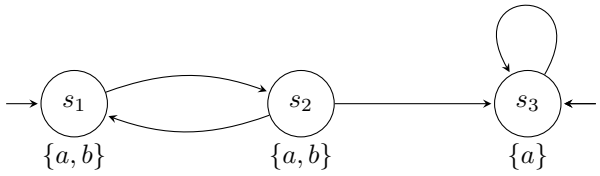

Figure 4.2. *Système de transition dont les traces peuvent être décrites par des formules LTL.*

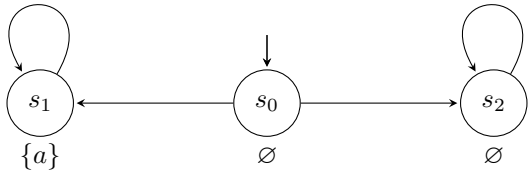

Figure 4.3. *Système de transition ST tel que* $ST \not\models \Diamond a$ *et* $ST \not\models \neg\Diamond a$.

Remarque (Spécification de propriétés temporisées). Pour décrire le comportement de systèmes réactifs au cours du temps, on peut considérer les itérations successives de l'opérateur temporel $\bigcirc$, en posant :

$$\bigcirc^k \varphi = \underbrace{\bigcirc \bigcirc \cdots \bigcirc}_{k \text{ fois}} \varphi,$$

où φ est une propriété LTL et k un entier naturel non nul. La formule $\bigcirc^k \varphi$ signifie alors « φ *est vraie après (exactement)* k *pas de temps* ».

On étend également la portée des opérateurs temporels $\Diamond$ et $\Box$ en posant :

$$\Diamond^{\leq k} \varphi = \bigvee_{0 \leq i \leq k} \bigcirc^i \varphi, \quad \Box^{\leq k} \varphi = \bigwedge_{0 \leq i \leq k} \bigcirc^i \varphi.$$

La formule $\Diamond^{\leq k} \varphi$ signifie alors « φ *sera vraie à un moment, dans les* k *pas de temps suivant l'instant présent* », et la formule $\Box^{\leq k} \varphi$ signifie « φ *sera toujours vraie pendant les* k *pas de temps suivant l'instant présent* ». ◁

4.1.3 Équivalence de formules LTL

Soit *Prop* un ensemble de propositions atomiques. On souhaite ici déterminer des règles d'équivalence de formules LTL sur *Prop*.

Définition 4.4 (Équivalence de formules LTL)**.** Deux formules LTL φ, ψ sur *Prop* sont dites équivalentes si les propriétés linéaires qu'elles induisent vérifient

$$Mots(\varphi) = Mots(\psi).$$

Dans ce cas, on note $\varphi \equiv \psi$. □

Puisque la logique LTL étend la logique propositionnelle, les équivalences de la logique propositionnelle sont aussi vérifiées pour la logique LTL. On a par exemple $\neg\neg\varphi \equiv \varphi$ et $\varphi \wedge \varphi \equiv \varphi$, pour toute formule LTL φ.

On peut montrer de nombreuses lois d'équivalence de formules LTL faisant intervenir les opérateurs temporels $\bigcirc$, $\Diamond$ et $\Box$. On a notamment les lois de distributivité suivantes :

$$\bigcirc(\varphi \,\mathcal{U}\, \psi) \equiv (\bigcirc\varphi)\,\mathcal{U}\,(\bigcirc\psi),$$
$$\Diamond(\varphi \vee \psi) \equiv (\Diamond\varphi) \vee (\Diamond\psi),$$
$$\Box(\varphi \wedge \psi) \equiv (\Box\varphi) \wedge (\Box\psi),$$

pour toutes formules LTL φ, ψ. Les lois de distributivité précédentes sont analogues aux lois de distributivité des symboles $\exists$ et $\vee$, ou $\forall$ et $\wedge$ dans le calcul des prédicats.

On doit prendre garde que l'opérateur temporel $\Diamond$ n'est pas distributif par rapport à l'opérateur de conjonction $\wedge$, et que l'opérateur temporel $\Box$ n'est pas distributif par rapport à l'opérateur de disjonction $\vee$:

$$\Diamond(\varphi \wedge \psi) \not\equiv (\Diamond\varphi) \wedge (\Diamond\psi), \quad \Box(\varphi \vee \psi) \not\equiv (\Box\varphi) \vee (\Box\psi).$$

Le système de transition illustré sur la figure 4.4 permet de montrer la première affirmation pour des propositions atomiques.

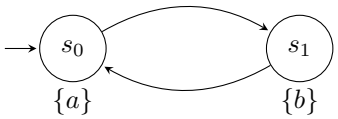

Figure 4.4. *Système de transition ST sur $\{a, b\}$ tel que $ST \not\models \Diamond(a \wedge b)$ et $ST \models \Diamond a \wedge \Diamond b$.*

4.1.4 Le problème du Model-Checking LTL

Soit *Prop* un ensemble de propositions atomiques, ST un système de transition et φ une formule LTL sur *Prop*. Le problème du Model-Checking LTL est un problème de décision : a-t-on *oui* ou *non* $ST \models \varphi$? Si $ST \not\models \varphi$, peut-on de plus obtenir un contrexemple sous la forme d'une trace de ST qui ne satisfait pas φ ? Pour un système de transition de grande taille, une étude manuelle n'est pas envisageable ; d'où la nécessité d'une méthode algorithmique qui peut être rendue automatique.

Supposons que le système de transition ST soit fini et sans état final. Pour vérifier si le système de transition ST satisfait la formule LTL φ, on exécute l'algorithme 4, dont la structure est très proche de l'algorithme 3 de vérification d'une propriété de sûreté régulière.

Algorithme 4 Algorithme du Model-Checking LTL

Entrée : système de transition fini ST et formule LTL φ sur Prop.
Sortie : « OUI » si ST satisfait φ, autrement « NON » et un contre-exemple.

Déterminer un automate de Büchi $\mathcal{B}$ tel que $\mathcal{L}_\omega(\mathcal{B}) = \mathit{Mots}(\neg\varphi)$
Construire le système de transition $ST \otimes \mathcal{B}$

Si il existe un chemin π dans $ST \otimes \mathcal{B}$ qui satisfait la condition d'acceptance de $\mathcal{B}$ **alors**
  renvoyer (NON, π)
Sinon
  renvoyer OUI
Fin Si

Dans l'algorithme 4, la première étape consiste à construire un automate de Büchi $\mathcal{B}$ dont le langage infini $\mathcal{L}_\omega(\mathcal{B})$ coïncide avec la propriété $\mathit{Mots}(\neg\varphi)$ induite par la formule LTL φ. L'existence et la construction d'un tel automate sont assurées par le théorème suivant, dont la preuve est admise.

Théorème 4.1. Pour toute formule LTL φ sur *Prop*, il existe un automate de Büchi $\mathcal{B}$ tel que $\mathcal{L}_\omega(\mathcal{B}) = \mathit{Mots}(\varphi)$, qui peut être construit avec une complexité de l'ordre de $2^{\mathcal{O}(|\varphi|)}$ en temps et en espace, où $|\varphi|$ est une mesure la taille de φ.

Exemple 4.4. Soit a une proposition atomique et $\mathit{Prop} = \{a\}$. L'automate illustré sur la figure 4.5 est un automate de Büchi dont le langage infini coïncide avec la propriété induite par la formule LTL $\Diamond\Box a$. •

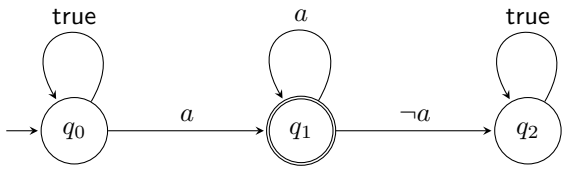

Figure 4.5. *Automate de Büchi dont le langage coïncide avec la propriété induite par la formule LTL* $\Diamond\Box a$.

Le théorème suivant établit finalement la classe de complexité du problème de décision du Model-Checking LTL. Une étape importante de sa démonstration, qui est admise, consiste à prouver que le problème du chemin Hamiltonien est polynomialement réductible au complémentaire du problème de décision du Model-Checking LTL (un chemin hamiltonien d'un graphe est un chemin qui passe par tous ses sommets une fois et une seule ; le problème du chemin hamiltonien est un problème de décision NP-complet qui consiste, étant donné un graphe, à décider s'il admet un chemin hamiltonien).

Théorème 4.2. Le problème du Model-Checking LTL est PSPACE-complet.

Rappelons que PSPACE désigne la classe des problèmes de décision qui peuvent être résolus par un algorithme déterministe polynomial en espace ; PTIME correspond aux problèmes de décision qui peuvent être résolus par un algorithme déterministe polynomial en temps ; NP correspond aux problèmes de décision qui peuvent être résolus par un algorithme non-déterministe polynomial en temps. On a de plus

$$\text{PTIME} \subseteq \text{NP} \subseteq \text{PSPACE}.$$

On ne sait toujours pas en 2022 si PTIME = NP ou si PTIME $\subsetneq$ NP (cette question constitue un des sept fameux *problèmes du millénaire*). Enfin, un problème de décision P est dit PSPACE-difficile si tout problème de décision Q dans PSPACE peut être réduit polynomialement à P ; on dit que P est PSPACE-complet si P appartient à PSPACE et P est PSPACE-difficile. Pour plus de détails sur ce sujet, on pourra consulter [10].

Remarque. La logique LTL repose sur une vision linéaire du déroulement temporel. Il existe néanmoins de nombreux systèmes logiques autres que la logique LTL. Par exemple, la logique CTL (*Computation Tree Logic* en anglais) permet de décrire des propriétés de systèmes de transition et repose sur une vision arborescente du déroulement temporel. Certaines logiques, comme PLTL et PCTL, ajoutent une dimension probabiliste. D'autres logiques, par exemple la logique CSL [1], ont été proposées afin de décrire les propriétés de systèmes stochastiques continus. ◁

4.2 Logique BLTL

La logique BLTL a été proposée très récemment (en 2019) par Liu, Gyori et Thiagarajan [12] ; cette logique temporelle est une adaptation de la logique LTL qui permet de spécifier certaines propriétés de trajectoires continues ; ces trajectoires, observées sur un intervalle de temps borné, décrivent l'évolution de systèmes continus, issus notamment des sciences du vivant ; elles peuvent par exemple être déterminées par des équations différentielles, par des processus probabilistes, ou encore par des modèles hybrides associant les deux formalismes.

4.2.1 Trajectoires continues

Considérons un système issu du vivant, dont l'évolution est observée sur un intervalle de temps borné $[0, T]$ avec $T > 0$. Ce système dépend de certains paramètres et définit des trajectoires continues, que nous supposons entièrement déterminées par les courbes représentatives de fonctions continues, définies sur $[0, T]$, à valeurs dans $\mathbb{R}$. Notons *Traj* l'ensemble de ces trajectoires ; on suppose que si

$\sigma \in \mathit{Traj}$, alors σ est la courbe représentative d'une fonction continue, également notée σ, définie sur $[0, T]$ à valeurs dans $\mathbb{R}$.

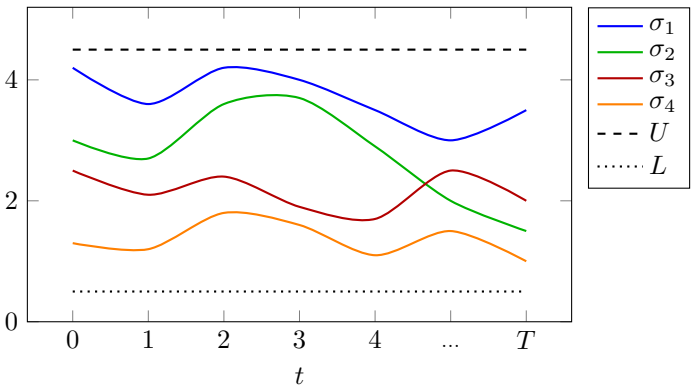

Figure 4.6. *Quelques trajectoires continues décrivant l'évolution d'un système issu du vivant.*

On suppose de plus que l'ensemble *Traj* est un ensemble borné, c'est-à-dire qu'il existe deux constantes réelles $L < U$ telles que pour toute trajectoire $\sigma \in \mathit{Traj}$, on ait

$$L \leq \sigma(t) \leq U, \quad \forall t \in [0, T].$$

Quelques trajectoires sont illustrées sur la figure 4.6. Elles peuvent par exemple modéliser la croissance d'une population d'individus biologiques, la propagation d'un caractère au sein d'une espèce, la concentration d'une substance chimique, etc.

4.2.2 Syntaxe et sémantique de la logique BLTL

Pour définir la syntaxe et la sémantique de la logique BLTL, on construit d'abord une discrétisation $\mathcal{T}$ de l'intervalle $[0, T]$. Pour simplifier, on suppose que T est entier et on pose

$$\mathcal{T} = \{0, 1, 2, 3, \ldots, T\}. \tag{4.1}$$

On considère ensuite un ensemble fini $\mathit{Prop} \subset \mathbb{R}^2$ de n propositions atomiques de la forme (l, u) avec $L \leq l < u \leq U$ (n est un entier naturel non nul). On note alors

$$\mathit{Prop} = \big\{(l_i, u_i), 1 \leq i \leq n\big\}. \tag{4.2}$$

Syntaxe

Définition 4.5 (Syntaxe BLTL)**.** Soit $\mathcal{T}$ l'ensemble d'instants défini par (4.1) et *Prop* l'ensemble de propositions atomiques défini par (4.2). Les formules BLTL sur *Prop* sont définies par les quatre règles suivantes :

- true est une formule BLTL,
- toute proposition atomique $(l, u) \in \mathit{Prop}$ est une formule BLTL,
- si φ est une formule BLTL, alors $\neg\varphi$ est aussi une formule BLTL,
- si φ_1, φ_2 sont des formules BLTL, alors $\varphi_1 \mathcal{U}^{\leq t} \varphi_2$ et $\varphi_1 \mathcal{U}^{t} \varphi_2$ sont aussi des formules BLTL, pour tout $t \in \mathcal{T}$.

□

Pour $k \in \mathcal{T}$ et φ une propriété BLTL, on définit également les formules BLTL $\Diamond^k\varphi$, $\Diamond^{\leq k}\varphi$ et $\Box^{\leq k}\varphi$:

$$\Diamond^k\varphi = \mathsf{true}\mathcal{U}^k\varphi, \quad \Diamond^{\leq k}\varphi = \mathsf{true}\mathcal{U}^{\leq k}\varphi, \quad \Box^{\leq k}\varphi = \neg\Diamond^{\leq k}\neg\varphi.$$

En particulier, la formule $\Diamond^k\varphi$ permet d'accéder à φ à l'instant $k \in \mathcal{T}$.

Sémantique

On définit ensuite la sémantique de la logique BLTL. L'interprétation d'une proposition atomique (l, u) pour une trajectoire σ à l'instant t s'énonce de la façon suivante : « *à l'instant présent, la valeur de* σ *est comprise entre* l *et* u » (voir figure 4.7).

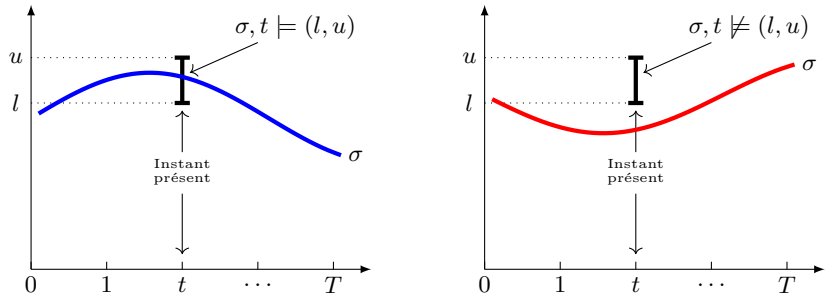

Figure 4.7. *Interprétation d'une proposition atomique pour la logique BLTL.*

Définition 4.6 (Sémantique BLTL)**.** La relation de satisfaction $(\sigma, t) \models \varphi$, où $\sigma \in \mathit{Traj}$, $t \in \mathcal{T}$ et φ est une formule BLTL est définie par les règles suivantes :

$$
\begin{array}{lcl}
(\sigma,t) \models \mathsf{true}, & & \\
(\sigma,t) \models (l,u) & \Leftrightarrow & l \leq \sigma(t) \leq u, \\
(\sigma,t) \models \varphi_1 \wedge \varphi_2 & \Leftrightarrow & (\sigma,t) \models \varphi_1 \text{ et } (\sigma,t) \models \varphi_2, \\
(\sigma,t) \models \neg\varphi & \Leftrightarrow & (\sigma,t) \not\models \varphi, \\
(\sigma,t) \models \varphi_1 \mathcal{U}^{\leq k} \varphi_2 & \Leftrightarrow & \exists k' \leq k \text{ tel que } t+k' \leq T, (\sigma,t+k') \models \varphi_2, (\sigma,t+k'') \models \varphi_1, \forall k'' \in [0,k'[, \\
(\sigma,t) \models \varphi_1 \mathcal{U}^{k} \varphi_2 & \Leftrightarrow & (\sigma,t+k) \models \varphi_2, (\sigma,t+k') \models \varphi_1, \forall k' \in [0,k[.
\end{array}
$$

Enfin, si φ est une formule BLTL, on définit

$$\mathit{models}(\varphi) = \big\{\sigma \in \mathit{Traj} \mid (\sigma, 0) \models \varphi\big\}.$$

□

Exemple 4.5 (Adéquation à des données d'observation)**.** On considère encore un système issu du vivant. On suppose qu'on dispose d'un modèle qui décrit son comportement au cours du temps, et qui détermine comme précédemment un ensemble de trajectoires *Traj*. On suppose de plus qu'on dispose d'un ensemble $\Gamma = \{\gamma_0, \gamma_1, \ldots, \gamma_T\}$ de données d'observation de ce système, relevées à chaque instant de l'ensemble $\mathcal{T}$. On construit alors un tunnel θ_α de rayon α autour de ces données :

$$\theta_\alpha = [\gamma_0 - \alpha, \gamma_0 + \alpha] \times [\gamma_1 - \alpha, \gamma_1 + \alpha] \times \cdots \times [\gamma_T - \alpha, \gamma_T + \alpha],$$

et on s'intéresse à la propriété φ_α : « *la trajectoire* σ *est toujours dans le tunnel* θ_α ».

La logique BLTL permet d'exprimer la propriété φ_α de la façon suivante :

$$\varphi_\alpha = \bigwedge_{0 \leq k \leq T} \Diamond^k (\gamma_k - \alpha, \gamma_k + \alpha).$$

La figure 4.8 montre deux trajectoires σ_1, σ_2 et un ensemble de données Γ ; on constate que $\sigma_1 \models \varphi_\alpha$ alors que $\sigma_2 \not\models \varphi_\alpha$.

•

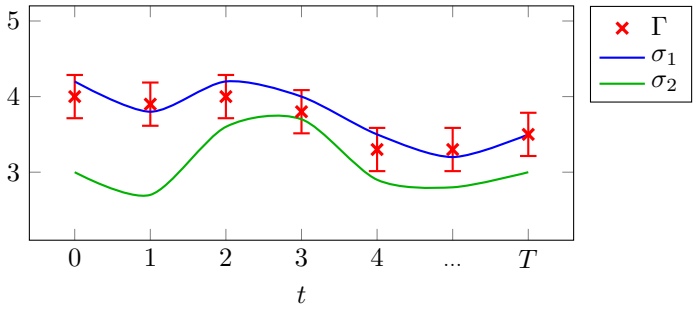

Figure 4.8. *Propriété d'adéquation d'une trajectoire à des données d'observation.*

4.2.3 Vérification statistique de propriétés

La vérification des propriétés de certains systèmes est parfois difficile à effectuer. Plutôt que de rechercher si une propriété donnée est satisfaite ou non, on peut alors calculer ou *estimer* la probabilité que cette propriété soit satisfaite.

Considérons donc un système qui détermine un ensemble de trajectoires *Traj* sur un intervalle de temps du type (4.1), et soit ψ une propriété BLTL sur un ensemble fini de propositions atomiques *Prop* de la forme (4.2). On supose que les formules BLTL forment un ensemble de probabilité, et on note $p = \mathbb{P}(\psi)$ la probabilité qu'une trajectoire $\sigma \in \mathit{Traj}$ vérifie la propriété ψ. Pour estimer la valeur de p, on peut appliquer une méthode de Monte-Carlo. Pour cela, on réalise N simulations indépendantes du système considéré. Chaque simulation produit une trace $\sigma_i \in \mathit{Traj}$, $1 \leq i \leq N$, qui vérifie ou non la propriété ψ ; on note X_i la variable qui prend pour valeur 1 si $\sigma_i \models \psi$, et la valeur 0 si $\sigma_i \not\models \psi$. Le théorème suivant donne une estimation par intervalle de confiance de la probabilité p.

Théorème 4.3. Soient $\varepsilon > 0$ et $\delta \in]0,1[$. On note $Y = \dfrac{1}{N} \displaystyle\sum_{1 \leq i \leq N} X_i$. Si $N \geq 4 \log\left(\frac{2}{\delta}\right) / \varepsilon^2$, alors on a

$$\mathbb{P}\big(|Y - p| \leq \varepsilon \big) \geq \delta.$$

La démonstration de ce théorème est présentée dans [8]. Sa mise en œuvre constitue une méthode de vérification statistique appelée *Statistic Model Checking*, qui repose sur la possibilité de simuler le système étudié afin d'en produire une trace, puis d'attribuer à cette trace un *score*, égal à 0 ou à 1 selon qu'elle vérifie ou non la propriété ψ considérée. Le coefficient δ représente le risque d'erreur dans le procédé d'estimation, et le coefficient ε représente la précision de l'estimation. Par exemple, si l'on souhaite une précision de 10% avec un risque d'erreur de 10%, il suffit de réaliser $N = 1200$ simulations indépendantes du système. En augmentant le nombre N de simulations, on obtient plus de précision ε ou on diminue le risque d'erreur δ.

Exemple 4.6. La méthode de vérification statistique qui repose sur le théorème 4.3 a été appliquée très récemment dans [16] pour réaliser une estimation de paramètres sur un modèle de croissance d'espèce océanique. •

4.3 Exercices

Exercice 1

1. Donner la sémantique des formules LTL $\Diamond\Box\varphi$ et $\Box\Diamond\varphi$.

2. Montrer les lois de dualité LTL suivantes :

$$\neg \bigcirc \varphi \equiv \bigcirc \neg\varphi,$$
$$\neg\Diamond\varphi \equiv \Box\neg\varphi,$$
$$\neg\Box\varphi \equiv \Diamond\neg\varphi.$$

3. Montrer les lois de distributivité LTL suivantes :

$$\bigcirc(\varphi \,\mathcal{U}\, \psi) \equiv (\bigcirc\varphi)\,\mathcal{U}\,(\bigcirc\psi),$$
$$\Diamond(\varphi \vee \psi) \equiv (\Diamond\varphi) \vee (\Diamond\psi),$$
$$\Box(\varphi \wedge \psi) \equiv (\Box\varphi) \wedge (\Box\psi).$$

4. Montrer la loi d'expansion LTL suivante :

$$\varphi \,\mathcal{U}\, \psi \equiv \psi \vee \big(\varphi \wedge \bigcirc(\varphi \,\mathcal{U}\, \psi)\big).$$

   En déduire des lois d'expansion similaires pour $\Diamond\varphi$ et $\Box\varphi$.

Exercice 2

On considère le système de transition ST_{sem} représentant l'algorithme d'exclusion mutuelle par sémaphore de deux processus P_1, P_2, admettant des états d'attente pause_1, pause_2 et des états critiques crit_1, crit_2 respectivement.

Ce système de transition vérifie-t-il les formules LTL suivantes ?

$$\Box(\neg\mathsf{crit}_1 \vee \neg\mathsf{crit}_2),$$
$$(\Box\Diamond\mathsf{crit}_1) \vee (\Box\Diamond\mathsf{crit}_2),$$
$$(\Box\Diamond\mathsf{crit}_1) \wedge (\Box\Diamond\mathsf{crit}_2),$$
$$(\Box\Diamond\mathsf{pause}_1 \to \Box\Diamond\mathsf{crit}_1) \wedge (\Box\Diamond\mathsf{pause}_2 \to \Box\Diamond\mathsf{crit}_2).$$

Exercice 3

On considère le système de transition sur l'ensemble de propositions atomiques $Prop = \{a, b\}$ représenté sur la figure 4.9 ci-dessous.

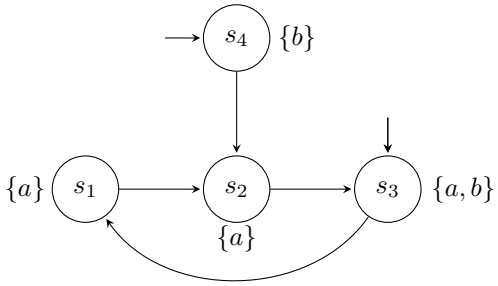

Figure 4.9. *Un système de transition sur l'ensemble de propositions atomiques* $Prop = \{a, b\}$.

Indiquer l'ensemble des états vérifiant les formules LTL suivantes :

(1) $\bigcirc a$,
(2) $\bigcirc\bigcirc\bigcirc a$,
(3) $\Box b$,
(4) $\Box\Diamond a$,
(5) $\Box(b \,\mathcal{U}\, a)$,
(6) $\Diamond(a \,\mathcal{U}\, b)$.

Exercice 4

L'opérateur LTL *weak until*, noté $\mathcal{W}$, est défini par

$$\varphi \mathcal{W} \psi = (\varphi \mathcal{U} \psi) \vee \Box \varphi.$$

1. Montrer que l'opérateur $\mathcal{W}$ vérifie les propriétés suivantes :

$$\neg(\varphi \mathcal{U} \psi) \equiv (\varphi \wedge \neg\psi) \mathcal{W} (\neg\varphi \wedge \neg\psi),$$
$$\neg(\varphi \mathcal{W} \psi) \equiv (\varphi \wedge \neg\psi) \mathcal{U} (\neg\varphi \wedge \neg\psi).$$

2. Déterminer une loi d'expansion pour l'opérateur $\mathcal{W}$.
3. Montrer que $\mathcal{W}$ est la solution la moins fine de cette loi d'expansion.
4. Quelle en est la solution la plus fine ?

Exercice 5

On considère un système (S) qui détermine un ensemble borné de trajectoires *Traj*, définies sur un intervalle $[0, T]$ avec T entier. On discrétise cet intervalle en posant $\mathcal{T} = \{0, 1, 2, 3, \ldots, T\}$.

Soit $m > 0$. Exprimer en logique BLTL les propriétés suivantes :

1. les trajectoires de (S) restent toujours dans un intervalle de rayon $\delta > 0$ autour de 0 ;
2. les trajectoires de (S) restent toujours dans un intervalle de rayon $\delta' > 0$ autour de m ;
3. les trajectoires de (S) sont attirées dans un intervalle de rayon $\delta' > 0$ autour de m ;
4. les trajectoires de (S) sont attirées définitivement dans un intervalle de rayon $\delta' > 0$ autour de m ;
5. les trajectoires de (S) oscillent entre un intervalle de rayon $\delta > 0$ autour de 0 et un intervalle de rayon $\delta' > 0$ autour de m.

Bibliographie

Note. L'essentiel de ce cours d'introduction au Model-Checking est issu de l'excellent livre de Christel Baier & Joost-Pieter Katoen [3].

[1] Adnan Aziz, Kumud Sanwal, Vigyan Singhal, and Robert Brayton. Verifying continuous time Markov chains. In *International Conference on Computer Aided Verification*, pages 269–276. Springer, 1996.

[2] Christel Baier, Boudewijn Haverkort, Holger Hermanns, and Joost-Pieter Katoen. Model checking continuous-time Markov chains by transient analysis. In *International Conference on Computer Aided Verification*, pages 358–372. Springer, 2000.

[3] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT press, 2008.

[4] E Clarke, Orna Grumberg, and David Long. Model checking. In *NATO ASI DPD*, pages 305–349, 1996.

[5] Edmund M Clarke. The birth of model checking. In *25 Years of Model Checking*, pages 1–26. Springer, 2008.

[6] Edmund M Clarke, E Allen Emerson, and Joseph Sifakis. Model checking : algorithmic verification and debugging. *Communications of the ACM*, 52(11) :74–84, 2009.

[7] Edmund M Clarke, Orna Grumberg, and David E Long. Model checking and abstraction. *ACM transactions on Programming Languages and Systems (TOPLAS)*, 16(5) :1512–1542, 1994.

[8] Thomas Hérault, Richard Lassaigne, Frédéric Magniette, and Sylvain Peyronnet. Approximate probabilistic model checking. In *International Workshop on Verification, Model Checking, and Abstract Interpretation*, pages 73–84. Springer, 2004.

[9] Stephen C Kleene et al. Representation of events in nerve nets and finite automata. *Automata studies*, 34 :3–41, 1956.

[10] Bernhard Korte, Jens Vygen, Jean Fonlupt, and Alexandre Skoda. *Optimisation combinatoire*. 2010.

[11] Axel Legay, Benoît Delahaye, and Saddek Bensalem. Statistical model checking : An overview. In *International conference on runtime verification*, pages 122–135. Springer, 2010.

[12] Bing Liu, Benjamin M Gyori, and PS Thiagarajan. Statistical model checking-based analysis of biological networks. In *Automated Reasoning for Systems Biology and Medicine*, pages 63–92. Springer, 2019.

[13] Robert McNaughton and Hisao Yamada. Regular expressions and state graphs for automata. *IRE transactions on Electronic Computers*, (1) :39–47, 1960.

[14] Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science*, pages 46–57. IEEE, 1977.

[15] Jean-Pierre Queille and Joseph Sifakis. Specification and verification of concurrent systems in CESAR. In *International Symposium on programming*, pages 337–351. Springer, 1982.

[16] Simon Ramondenc, Damien Eveillard, Lionel Guidi, Fabien Lombard, and Benoit Delahaye. Probabilistic modeling to estimate jellyfish ecophysiological properties and size distributions. *Scientific Reports*, 10(1) :1–13, 2020.

[17] Davide Sangiorgi. *Introduction to bisimulation and coinduction*. Cambridge University Press, 2011.

[18] Ken Thompson. Programming techniques : Regular expression search algorithm. *Communications of the ACM*, 11(6) :419–422, 1968.