Sémantique et Applications à la Vérification de Programmes

Xavier Rival & Jérôme Féret

16 février 2024

Table des matières

1 Introduction

1.1 Études de Cas

1.1.1 Ariane 5, Vol 501, 1996

À son premier vol, la fusée Ariane 5, remplaçante de la Ariane 4 s'est désintégrée. Au bout de 36.7s elle a changé d'angle d'attaque, et au bout de 39s elle s'est désintégrée.
La fusée contenait des capteurs qui envoyaient leurs résultats à 2 calculateurs (un SRT et un OBC) puis aux moteurs. La fusée contenait des registres de $16, 32$ et 64 bits. Les opérations sur flottants se faisaient sur 64 bits et les entiers signés étaient stockés sur 16 bits. A la copie de donnée, des conversions sont faites avec des arrondis. Quand les valeurs sont trop grandes, on a soit une interruption du processus (code de gestion d'erreurs, ou crash) ou un comportement inattendu. Le SRI tournait en mode interruptif, mais n'avait pas de code de gestion d'erreurs. Ceci a causé à l'explosion :

- Conversion d'un flottant 64 bits vers un entier signé 16 bit qui cause un overflow
- Une interruption a lieu
- Le SRI crash par manque de code de gestion d'erreur

- Le crash envoie un code d'erreur qui est envoyé à l'OBC
- L'OBC interprète cette valeur comme des données de vol
- On calcule une trajectoire absurde

Pour éviter ça :

- Désactiver les interruptions sur les overflows : En cas de problème de capacité, on produit des valeurs incorrectes qui ne vont pas arrêter le calcul.
- Fixer le code du SRI de sorte qu'aucun overflow ne va avoir lieu, c'est ce qu'on appelle de la programmation défensive. Cela va être couteux à cause de nombreux tests :
- Gérer les erreurs de conversion de sorte que l'OBC soit au courant de ceci.

Le crash est aussi dû à des morceaux de code très anciens qui n'ont que peu été modifiés, et les hypothèses faites n'étaient plus vraies...
Le système était red !70 !blackondant matériellement, mais pas logiciellement, donc les deux SRI ont plantés. On peut pour éviter cela avoir deux sets indépendants de contrôle qui ont chacun trois unités de calcul.
De nos jours, on peut éviter de tels erreurs par des outils d'analyse statique.

1.1.2 Lufthansa Vol 2904, Varsovie, 1993

A l'atterissage à l'aéroport de varsovie, un airbus A320 de la Lufthansa s'est craché contre une colline à 100km/h. Les conditions météo étaient mauvaises, la piste était mouillée, l'atterrissage a été suivi d'aqua-planing et d'un freinage délayé.

- A l'origine, les conditions d'atterissage n'ont pas été correctement évaluées par l'équipage, le vent latéral était trop fort. L'équipage n'aurait pas dû essayer d'atterir.
- Mais en plus, le système de freinage a été retardé de 9s.

Les systèmes de freinage ont une fonctionnalité d'inhibition pour empêcher une activation en vol : les spoilers augmentent la charge aérodynamique et les inverseurs de poussée pourraient détruire l'avion si activé en vol. Selon le logiciel, il ne devrait pas s'activer à moins que le levier de poussée soit au minimum et que : ou bien la masse sur chacune des roues est d'au moins $6T$ ou bien les roues sont en train de tourner avec une vitesse d'au moins 130km/h.

A cause de l'aqua-planing, les roues n'ont pas tourné et à cause du vent, le train d'atterissage gauche ne portait pas de masse. Même si la commande de poussée était bien au minimum, ça ne suffisait pas.
En changeant la condition : $P_{left} > 6T \wedge P_{right} > 6T$ pour $P_{left} + P_{right} > 12T$. Cette solution ne peut être comprise qu'avec une connaissance de l'environnement.

1.1.3 Missile Patriot, Dahran, 1991

Le système Patriot était un système de défense anti-missile. IL a été utilisé dans la première guerre du Golfe avec un taux de succès de 50%. Les échecs sont dûs à un échec au lancement plus qu'à l'échec de la destruction.
Le système doit toucher des cibles se déplaçant très rapidement (1700m/s) sans toucher les cibles alliées et ce avec un coût énorme.

Un système de détection avec des radar est suivi d'une confirmation de trajectoire (vérification en $t+\delta = t+0,1s$ de la position) et d'une identification de la cible. On calcule alors une trajectoire d'interception et de lancement ultra précise. Le problème a eu lieu à cause de la confirmation de trajectoire.
Ici, un missile a été détecté mais le système a été incapable de confirmer la trajectoire du missile : il y a eu une imprécision dans le calcul de l'horloge, ce qui cause le calcul d'une mauvaise fenêtre de confirmation et l'échec de confirmation de la trajectoire.
L'erreur est liée à des arrondis causés par l'arithmétique à précision fixe et l'impossibilité de représenter exactement 1/10 dans celle ci. L'horloge matérielle dérive alors de $.34s$ toutes les $100h$ ce qui modifie la position de la fenêtre de confirmation de $580m$.

1.1.4 En Général

Les raisons habituelles pour les problèmes logiciels sont les suivantes :

- Une spécification ou compréhension de l'environnement inadaptée
- Une implémentation incorrecte de la spécification
- Une compréhension incorrecte du modèle d'exécution

On doit s'adapter à des architectures logiciel complexes par exemple les logiciels parallélélisés (un processeur multi-thread, plusieurs processeurs). Il est difficile de vérifier l'exécution de ces logiciels.
On doit également vérifier des propriétés complexes. Pour la sécurité par exemple, le système doit résister en présence d'un attaquant, et si des données sensibles sont touchées, ou des données critiques sont corrompues la vérification n'est pas suffisante. Les propriétés de sécurité sont difficiles à exprimer.

On a alors des techniques pour confirmer la sécurité logicielle :

- Du côté logiciel :
  — Porter une attention à la spécification et à la qualité logicielle
  — Fixer des règles de programmations

  Ceci ne garantit généralement aucune propriété forte, mais aide à la vérification.
- Du côté formel :
  — Cela nécessite des fondations mathématiques sûres.
  — Ceci doit permettre de garantir que des logiciels vérifient des propriétés complexes.
  — Il faut pouvoir y faire confiance, comment être sûr que la preuve d'un article est juste ?

En résumé :

Définition 1.1: Sémantique et Vérification

Sémantique
- Permet de décrire précisément le comportement des programmes
- Permet d'exprimer les propriétés à vérifier
- Est utile pour transformer et compiler des programmes

Vérification
- Cherche à prouver des propriétés sémantiques des programmes
- Est indécidable et donc se résout à faire des compromis variés selon plusieurs approches.

1.2 Approches de la Vérification

1.2.1 Indécidabilité et Limitations Fondamentales

Définition 1.2: Arrêt

Un programme P termine sur une entrée X si et seulement si toute exécution de P sur l'entrée X atteint un état final.

Théorème 1.1: Indécidabilité

Le problème de l'Arrêt est indécidable. De même, l'absence d'erreur à l'exécution est indécidable.

Plus généralement :

Définition 1.3: Spécification Sémantique

Une spécification sémantique est un ensemble d'exécutions correctes de programmes.

Théorème 1.2: Rice

Étant donné un langage Turing-Complet, toute spécification sémantique non-triviale est indécidable.

1.2.2 Approches à la Vérification

On va souvent calculer les solutions à un problème plus faible que la vérification :

Simulation/Tests On observe un nombre fini d'exécutions finies.

Preuves Assistée On abandonne l'automatisation

Vérification de Modèles On ne considère que des sytèmes finis

Recherche de Bug On recherche des Patterns qui indiquent des erreurs probables.

Analyse Statique Abstraite On cherche quand même à automatiser les preuves de correction en acceptant d'échouer sur des programmes corrects.

Définition 1.4: Vérification de la sûreté d'un Problème

Sémantique d'un Programme On définit $[\![P]\!]$ la sémantique d'un programme P comme l'ensemble des comportements de P (i.e. ses états)

Propriété à Vérifier On définit une propriété à vérifier $\mathcal{S}$ un ensemble de comportements admissibles (i.e. des états sécures)

L'objectif est alors de vérifier si $[\![P]\!] \subseteq \mathcal{S}$.

Définition 1.5: Propriétés d'un Vérificateur

Automatisation Existence d'un Algorithme

Passage à l'échelle Capacité à gérer des logiciels lourds

Robustesse Doit identifier tous les programmes défaillants

Complétude Doit accepter tous les programmes corrects

Application au Source Ne requière pas de phase de modélisation

Test Par Simulation On exécute le programme sur un nombre fini d'entrées finies. Cette méthode est très utilisée, automatisées, complète mais coûteuse et non sûre.

Preuve Assistée Par Ordinateur (PAO) On vérifie par la machine une preuve en partie écrite par l'humain. Cette méthode est souvent appliquée, pas complètement automatisée, sûre et quasi-complète (du moins en pratique)

Vérification de Modèles On ne considère que systèmes finis, en utilisant des algorithmes pour l'exploration exhaustive et des réductions par symétrie. Cette méthode est régulièrement appliquée au hardware et aux drivers, s'applique sur un modèle (ce qui nécessite une phase d'extraction du modèle), ce qui n'est pas toujours automatisable et qui est approché pour les systèmes infinis. C'est toutefois automatisé, sûr et complet par rapport au modèle.

Bug Finding On identifie des problèmes « probables »c'est à dire des patterns connus pour souvent amener à des erreurs. On utilise des exécutions symboliques bornées et on hiérarchise les problèmes avec des heuristiques. Cette méthode est automatisable, incomplète et non sûre.

Analyse Statique On utilise des approximations d'une manière conservatrice :

- Sous-approximation de la propriété : $\mathcal{S}_{\text{under}} \subseteq \mathcal{S}$
- Sur-approximation de la sémantique : $[\![P]\!] \subseteq [\![P]\!]_{\text{upper}}$
- On calcule $[\![P]\!]_{\text{upper}}, \mathcal{S}_{\text{under}}$ et on vérifie si $[\![P]\!]_{\text{upper}} \subseteq \mathcal{S}_{\text{under}}$.

Cette méthode est sûre : l'abstraction attrapera tout programme incorrecte : $[\![P]\!] \not\subseteq \mathcal{S} \implies [\![P]\!]_{\text{upper}} \not\subseteq \mathcal{S}_{\text{under}}$.

Toutefois, cette méthode est incomplète, on peut avoir $[\![P]\!]_{\text{upper}}$ qui sort de $\mathcal{S}_{\text{under}}$ et même de $\mathcal{S}$.

Finalement :

	Automatisable	Sûr	Complet	Source Level	Scalable
Simulation	Oui	Non	Oui	Oui	Parfois
PAO	Non	Oui	Presque	Partiellement	Parfois
Modèle-Check	Oui	Oui	Partiellement	Non	Parfois
Bug-Finding	Oui	Non	Non	Oui	Parfois
Analyse Statique	Oui	Oui	Non	Oui	Parfois

1.3 Ordres, Treillis, Points Fixes

1.3.1 Relations d'Ordres

Définition 1.6: Poset

Une relation d'ordre $\sqsubseteq$ sur un ensemble $\mathcal{S}$ est une relation binaire $\sqsubseteq \subseteq \mathcal{S} \times \mathcal{S}$ sur $\mathcal{S}$ qui est réfléxive, transitive et antisymétrique. On notera $\sqsubset$ l'ordre strict associé à la relation d'ordre $\sqsubseteq$.

Proposition 1.1: Sémantique d'un Automate

On considère un automate $\mathcal{A} = (Q, q_i, q_f, \rightarrow)$ sur l'alphabet L. On note $\mathcal{L}[\mathcal{A}]$ le langage reconnu par l'automate. On peut alors définir des propriétés sémantiques sur le langage :

$\mathcal{P}_0$ Aucun mot reconnu ne contient deux b consécutifs :

$$\mathcal{L}[\mathcal{A}] \subseteq L^* \setminus L^* bbL^*$$

$\mathcal{P}_1$ Tous les mots reconnus contiennent au moins un a :

$$\mathcal{L}[\mathcal{A}] \subseteq L^* aL^*$$

$\mathcal{P}_2$ Les mots reconnus ne contiennent pas de b :

$$\mathcal{L}[\mathcal{A}] \subseteq (L \setminus \{b\})^*$$

Définition 1.7: Ordre Total

Un ordre est total si tous deux éléments sont comparables.

Définition 1.8: Éléments Extrémaux

Soit $\mathcal{S}' \subseteq (\mathcal{S}, \sqsubseteq)$. Alors x est :

- Un élément minimal de $\mathcal{S}'$ si et seulement si $x \in \mathcal{S}' \wedge \forall y \in \mathcal{S}', x \sqsubseteq y$
- Un élément maximal de $\mathcal{S}'$ si et seulement si $x \in \mathcal{S}' \wedge \forall y \in \mathcal{S}', y \sqsubseteq x$

Si $x \in \mathcal{S}$, on dit que x est un majorant de $\mathcal{S}'$ si

$$\forall y \in \mathcal{S}', y \sqsubseteq x$$

et une borne supérieure si de plus :

$$\forall y \in \mathcal{S}', y \sqsubseteq x \wedge \forall z \in \mathcal{S}, (\forall y \in \mathcal{S}', y \sqsubseteq z) \implies x \sqsubseteq z$$

On note alors $x = \sqcup \mathcal{S}'$. On a aussi des notions duales de minorant et de borne inférieure.

Définition 1.9: Treillis Complet

On appelle treillis complet un sextuplet $(\mathcal{S}, \sqsubseteq, \perp, \top, \sqcup, \sqcap)$ où :

- $\sqsubseteq$ est un ordre sur $\mathcal{S}$
- $\perp$ est l'infimum de $\mathcal{S}$
- $\top$ est le suprémum de $\mathcal{S}$
- Toute partie $\mathcal{S}'$ de $\mathcal{S}$ a une borne supérieure $\sqcup \mathbb{S}'$ et une borne inférieure $\sqcap \mathcal{S}'$.

Définition 1.10: Treillis

On appelle treillis un sextuplet $(\mathcal{S}, \sqsubseteq, \perp, \top, \sqcup, \sqcap)$ où :

- $\sqsubseteq$ est un ordre sur $\mathcal{S}$
- $\perp$ est l'infimum de $\mathcal{S}$
- $\top$ est le suprémum de $\mathcal{S}$
- Toute paire $\{x, y\}$ de $\mathcal{S}$ a une borne supérieure $x \sqcup y$ et une borne inférieure $x \sqcap y$.

Par exemple : $\mathcal{Q} = q \in \mathbb{Q} \mid 0 \leq q \leq 1$ est un treillis pour $\leq$ mais n'est pas complet, puisque $\{q \in \mathcal{Q} \mid q \leq \frac{\sqrt{2}}{2}\}$ n'a pas de borne supérieure dans $\mathcal{Q}$.
Un treillis fini est complet.

Définition 1.11: Chaine Croissante

Une partie $\mathcal{C}$ d'un poset $\mathcal{S}, \sqsubseteq$ est une chaîne croissante si :

- Elle a un infimum
- Le poset $\mathcal{C}, \sqsubseteq$ est total.

Le poset $\mathcal{S}, \sqsubseteq$ vérifie la condition de chaîne croissante si et seulement si toute chaîne croissante est finie.

Définition 1.12: Ordre Partiel Complet

Un ordre partiel complet (cpo) est un poset de sorte que toute chaîne croissante a au moins une borne supérieure. Un cpo pointé est un cpo avec un infimum $\perp$.

1.3.2 Opérations sur un Poset et Points-Fixes

On se donne un automate $\mathcal{A}$ et une propriété à vérifier $\mathcal{S}$. On veut prouver $[\![\mathcal{A}]\!] \subseteq \mathcal{S}$ par induction, il faudrait donc pouvoir définir de manière constructive les sémantiques. Pour un automate, on observe :

Observation 1 $\mathcal{L}[\mathcal{A}] = [\![A]\!] (q_f)$ où

$$[\![A]\!] : q \mapsto \{w \in L^* \mid \text{il existe un calcul sur } w \text{ qui atteint } q\}$$

Observation 2 $[\![\mathcal{A}]\!] = \bigcup_{n \in \mathbb{N}} [\![\mathcal{A}]\!]_n$ où

$$[\![\mathcal{A}]\!]_n : q \mapsto \{a_0 \dots a_{n-1} \mid \text{il existe un calcul sur } a_0 \dots a_{n-1} \text{ qui atteint } q\}$$

Observation 3 $[\![\mathcal{A}]\!]_{n+1}$ se calcule récursivement :

$$[\![\mathcal{A}]\!]_{n+1} (q) = \bigcup_{q' \in Q} \left\{ wa \mid w \in [\![\mathcal{A}]\!]_n (q') \wedge q' \xrightarrow{a} q \right\}$$

On aurait aussi pu calculer $[\![A]\!]$ en notant $[\![A]\!]_n (q)$ les mots reconnus de longueurs n, et en calculant le résultat en prenant l'union.
On va dans la suite étudier une manière de définir les sémantiques par des points fixes.

Définition 1.13: Opérateurs

Soit φ un opérateur sur un poset $\mathcal{S}$:

- φ est croissant si et seulement si $x \sqsubseteq y \Longrightarrow \varphi(x) \sqsubseteq \varphi(y)$.
- φ est continu si et seulement si pour toute chaine $\mathcal{S}' \subseteq \mathcal{S}$ alors : Si $\sqcup\mathcal{S}'$ existe, alors $\sqcup\varphi(\mathcal{S}')$ aussi et $\varphi(\sqcup\mathcal{S}') = \sqcup\varphi(\mathcal{S}')$
- $\sqcup$-préservant si et seulement si, pour toute partie $\mathcal{S}' \subseteq \mathcal{S}$ alors : Si $\sqcup\mathcal{S}'$ existe, alors $\sqcup\varphi(\mathcal{S}')$ aussi et $\varphi(\sqcup\mathcal{S}') = \sqcup\varphi(\mathcal{S}')$

Proposition 1.2: Propriétés

- Si φ est continu, il est aussi croissant.
- Si φ préserve $\sqcup$, il est aussi croissant.

Démonstration. On suppose que φ préserve $\sqcup$, $x, y \in \mathcal{S}$ tels que $x \sqsubseteq y$. Alors, $\{x, y\}$ est une chaîne de borne supérieure y donc $\varphi(x) \sqcup \varphi(y)$ existe et est égal à $\varphi(y)$. Donc $\varphi(x) \sqsubseteq \varphi(y)$. ■

Définition 1.14: Points Fixes

- Un point fixe de φ est un élément x tel que $\varphi(x) = x$.
- Un pré point fixe de φ vérifie $x \sqsubseteq \varphi(x)$
- Un post-point fixe de φ vérifie $x \sqsupseteq \varphi(x)$
- Le plus petit point fixe $\operatorname{lfp}\varphi$ de φ (s'il existe, est unique) est le plus petit point fixe de φ. De même pour son plus grand point fixe $\operatorname{gfp}\varphi$.

Théorème 1.3: Tarski

Soit $\mathcal{S}$ un treillis complet et φ un opérateur croissant sur $\mathbb{S}$. Alors :

1. $\operatorname{lfp}\varphi = \sqcap\{x \in \mathcal{S} \mid \varphi(x) \sqsubseteq x\}$
2. $\operatorname{gfp}\varphi = \sqcup\{x \in \mathcal{S} \mid x \sqsubseteq \varphi(x)\}$
3. L'ensemble des points fixes de φ est un treillis complet.

Démonstration.

1. On pose $X = \{x \in \mathcal{S} \mid \varphi(x) \sqsubseteq x\}$ et $x_0 = \sqcap X$.
   Pour tout $y \in X$ on a :
   - $x_0 \sqsubseteq y$ par définition de la borne inférieure
   - $\varphi(x_0) \subseteq \varphi(y)$ par croissance de φ.
   - $\varphi(x_0) \subseteq y$ par défintion de X.

   Donc $\varphi(x_0) \sqsubseteq x_0 \sqsubseteq \varphi(x_0)$. Donc x_0 est un point fixe qui est une borne inférieure.
2. De même, par dualité.
3. Si X est un ensemble de points fixes de φ, on considère φ sur $\{y \in \mathcal{S} \mid y \sqsubseteq_{\mathcal{S}} \sqcap X\}$ pour établir l'existence d'une borne inférieure de X sur le poset des points fixes. L'existence d'une borne supérieure dans le poset des points fixes en découle par dualité.

■

Dans l'exemple des automates :

Treillis On prend $\mathcal{S} = Q \to \mathcal{P}(L^*)$ et comme ordre l'extension point à point $\sqsubseteq$ de $\subseteq$.

Opérateur On pose $\varphi_0 : \mathcal{S} \to \mathcal{S}$ défini par

$$\varphi_0(f) = \lambda(q \in Q) \cdot \bigcup_{q' \in Q} \left\{ wa \mid w \in f(q') \wedge q' \xrightarrow{a} q \right\}$$

On définit alors φ par :

$$\varphi(f) = \lambda(q \in Q) \cdot \begin{cases} f(q_i) \cup \varphi_0(f)(q_i) \cup \{\varepsilon\} & \text{si } q = q_i \\ f(q) \cup \varphi_0(f)(q) & \text{sinon} \end{cases}$$

Reste à Prouver L'existence de $\operatorname{lfp}\varphi$ découle du théorème de Tarski et l'égalité $\operatorname{lfp}\varphi = [\![\mathcal{A}]\!]$ peut être établie par induction et double inclusion, mais on peut faire ça plus simplement.

Théorème 1.4: Kleene

Soit $\mathcal{S}$ un cpo pointé et φ un opérateur continu sur $\mathcal{S}$. Alors φ a un plus petit point fixe and

$$\operatorname{lfp}\varphi = \bigsqcup_{n\in\mathbb{N}} \varphi^n(\bot)$$

Démonstration. Premièrement, on prouve l'existence de la borne supérieure :
Puisque φ est continu, il est aussi croissant. On prouve par induction sur n que $\{\varphi^n(\bot) \mid n \in \mathbb{N}\}$ est une chaîne :

- $\varphi^0(\bot) = \bot \sqsubseteq \varphi(\bot)$ par définition.
- Si $\varphi^n(\bot) \sqsubseteq \varphi^{n+1}(\bot)$ alors $\varphi^{n+1}(\bot) = \varphi(\varphi^n(\bot)) \sqsubseteq \varphi\left(\varphi^{n+1}(\bot)\right) = \varphi^{n+2}(\bot)$

Par définition, la borne supérieure existe donc. On la note x_0.

On doit maintenant prouver que c'est un point fixe de φ.
Puisque φ est continu, $\left\{\varphi^{n+1}(\bot) \mid n \in \mathbb{N}\right\}$ a une borne supérieure et :

$$\begin{aligned} \varphi(x_0) &= \varphi(\sqcup\{\varphi^n(\bot) \mid n \in \mathbb{N}\}) \\ &= \sqcup\left\{\varphi^{n+1}(\bot) \mid n \in \mathbb{N}\right\} && \text{par continuité de } \varphi \\ &= \bot \sqcup \left(\sqcup\left\{\varphi^{n+1}(\bot) \mid n \in \mathbb{N}\right\}\right) && \text{par définition de } \bot \\ &= x_0 && \text{par réécriture} \end{aligned}$$

Finalement, on doit montrer que c'est le plus petit point fixe.
Soit x_1 un point fixe de φ. On veut montrer par induction sur n que $\varphi^n(\bot) \sqsubseteq x_1$:

- $\varphi^0(\bot) \sqsubseteq x_1$ par définition de $\bot$.
- Si $\varphi^n(\bot) \sqsubseteq x_1$, par croissance de φ, $\varphi^{n+1}(\bot) \sqsubseteq \varphi(x_1) = x_1$.

Par définition de la borne supérieure, $x_0 \sqsubseteq x_1$. ■

On a maintenant une définition constructive de la sémantique des automates.
On a défini φ par

$$\varphi(f) = \lambda(q \in Q) \cdot \begin{cases} f(q_i) \cup \varphi_0(f)(q_i) \cup \{\varepsilon\} & \text{si } q = q_i \\ f(q) \cup \varphi_0(f)(q) & \text{sinon} \end{cases}$$

Puisque φ est continu, le théorème de Kleene s'applique donc $\operatorname{lfp}\varphi$ existe et $\operatorname{lfp}\varphi = \bigcup_{n\in\mathbb{N}} \varphi^n(\bot) = [\![A]\!]$.

On peut formaliser les définitions par récurrence de sémantiques :

Définition basée sur des règles d'inférence	Même propriété basée sur des points-fixes
$\dfrac{}{x_0 \in \mathcal{X}} \qquad \dfrac{x \in \mathcal{X}}{f(x) \in \mathcal{X}}$	$\operatorname{lfp}(Y \mapsto \{x_0\} \cup Y \cup \{f(x) \mid x \in Y\})$

Pour prouver l'inclusion d'un point fixe dans un set :

- Soit $\varphi : \mathcal{S} \to \mathcal{S}$ un opérateur continu
- Soit $\mathcal{I} \in \mathcal{S}$ tel que :
$$\forall x \in \mathcal{S}, x \sqsubseteq \mathcal{I} \Longrightarrow \varphi(x) \sqsubseteq \mathcal{I}$$
- On a $\bot \sqsubseteq \mathcal{I}$
- On peut prouver que $\operatorname{lfp} \varphi \sqsubseteq \mathcal{I}$.

2 Sémantique Opérationnelle

2.1 Systèmes de Transitions et Sémantique à Petit Pas

Définition 2.1: Système de Transition

Un système de transition est un tuple $(\mathbb{S}, \to)$ où :
- $\mathbb{S}$ est l'ensemble des états du système.
- $\to \subseteq \mathbb{S} \times \mathbb{S}$ est la relation de transition du système.

Un système est dit déterministe si un état détermine de manière unique le prochain état. Sinon, il est dit non déterministe. La relation $\to$ définit des étapes atomiques d'exécution : on appelle donc ce paradigme sémantique à petit pas. On ne considère pas de systèmes probabilistes.

Définition 2.2: États Particuliers

On considère souvent :
- Des états initiaux $\mathbb{S}_{\mathcal{I}} \subseteq \mathbb{S}$ qui dénotent des états de début d'exécution.
- Des états finaux $\mathbb{S}_{\mathcal{F}} \subseteq \mathbb{S}$ qui dénotent des états de fin du programme.
- Des états bloquants qui ne sont l'origine d'aucune transition. On introduit souvent un état d'erreur Ω pour les configurations bloquantes.

Par exemple, on rappelle les définitions du λ-calcul :

Définition 2.3: λ-termes

L'ensemble des λ-termes est défini par :

$$\begin{array}{llll} t, u, \ldots & ::= & x & \text{variable} \\ & \mid & \lambda x \cdot t & \text{abstraction} \\ & \mid & t\ u & \text{application} \end{array}$$

Définition 2.4: β-réduction

- $(\lambda x \cdot t)u \underset{\beta}{\to} t[x \leftarrow u]$
- Si $u \underset{\beta}{\to} v$ alors $\lambda x \cdot u \underset{\beta}{\to} \lambda x \cdot v$
- Si $u \underset{\beta}{\to} v$ alors $u\ t \underset{\beta}{\to} v\ t$
- Si $u \underset{\beta}{\to} v$ alors $t\ u \underset{\beta}{\to} t\ v$

2.2 Sémantiques de Traces

Définition 2.5: Traces d'Exécution

- Une trace finie est une suite finie d'état notée $\langle s_0, \ldots, s_n \rangle$
- Une trace infinie est une suite infinie d'état $\langle s_0, \ldots \rangle$
- On note $\mathbb{S}^*$ l'ensemble des traces finies et $\mathbb{S}^\omega$ l'ensemble des traces infinies et $\mathbb{S}^\alpha = \mathbb{S}^* \bigcup \mathbb{S}^\omega$ l'ensemble des traces.

Définition 2.6: Notations

On notera :

- ε la trace fide
- $|\cdot|$ l'opérateur de longueur
- $\cdot$ l'opérateur de concaténation
- $\prec$ la relation de préfixation

Définition 2.7: Sémantique de Traces Finies

La sémantique de traces finies $[\![\mathcal{S}]\!]^*$ est définie par

$$[\![\mathcal{S}]\!]^* = \{\langle s_0, \ldots, s_n \rangle \in \mathbb{S}^* \mid \forall i, s_i \to s_{i+1}\}$$

Définition 2.8: Parties Intéressantes des Sémantiques

- Les traces initiales commençant d'un état initial
- Les traces atteignant un état bloquant
- Les traces atteignant un état final
- Les traces maximales qui sont à la fois initiales et finales

On peut dériver les traces de longueur $i+1$ à partir des traces de longueur i :

Théorème 2.1: Définition par Points Fixe

n pose $\mathcal{I} = \{\varepsilon\} \cup \{\langle s \rangle \mid s \in \mathbb{S}\}$. On définit F_* par :

$$F_* : \begin{array}{lcl} \mathcal{P}(\mathbb{S}^*) & \longrightarrow & \mathcal{P}(\mathbb{S}^*) \\ X & \longmapsto & \mathcal{I} \cup \{\langle s_0, \ldots, s_n, s_{n+1} \rangle \mid \langle s_0, \ldots, s_n \rangle \in X \wedge s_n \to s_{n+1}\} \end{array}$$

Alors F_* est continue sur $\mathcal{P}(\mathbb{S}^*)$ et a donc un plus petit point fixe avec

$$\mathrm{lfp}\, F_* = \bigcup_{n \in \mathbb{N}} F_*^n(\varnothing) = [\![\mathbb{S}]\!]^*$$

Démonstration. On prouve d'abord que F_* est continue. On prend $\chi \subseteq \mathcal{P}(\mathbb{S}^*)$ telle que $\chi \neq \varnothing$ et $A = \bigcup_{U \in \chi} U$. Alors :

$$\begin{aligned} & F_*\left(\bigcup_{X \in \chi} X\right) \\ & \quad = \mathcal{I} \cup \left\{\langle s_0, \ldots, s_n, s_{n+1} \rangle \mid \left(\langle s_0, \ldots, s_n \rangle \in \bigcup_{U \in \chi} U\right) \wedge s_n \to s_{n+1}\right\} \\ & \quad = \mathcal{I} \cup \{\langle s_0, \ldots, s_n, s_{n+1} \rangle \mid \exists U \in \chi\, \langle s_0, \ldots, s_n \rangle \in U \wedge s_n \to s_{n+1}\} \\ & \quad = \mathcal{I} \cup \left(\bigcup_{U \in \chi} \{\langle s_0, \ldots, s_n, s_{n+1} \rangle \mid \langle s_0, \ldots, s_n \rangle \in U \wedge s_n \to s_{n+1}\}\right) \\ & \quad = \bigcup_{U \in \chi} (\mathcal{I} \cup \{\langle s_0, \ldots, s_n, s_{n+1} \rangle \mid \langle s_0, \ldots, s_n \rangle \in U \wedge s_n \to s_{n+1}\}) \\ & \quad = \bigcup_{U \in \chi} F_*(U) \end{aligned}$$

En particulier, si χ est une chaine, on démontre bien l'existence d'un plus petit point fixe par théorème de Kleene.
Ensuite, il reste à montrer que $[\![S]\!]^* = \mathrm{lfp}\, F_*$. Par induction sur n on a :

$$\forall n \geq 1, \forall k < n, \langle s_0, \ldots, s_k \rangle \in F_*^n(\varnothing) \Longleftrightarrow \langle s_0, \ldots, s_k \rangle \in [\![S]\!]^*$$

■