Rapport de projet I.R.L.

DELANAUX Rémy, ENSIMAG - 2A ISI

13 mai 2015

1 Introduction

1.1 Environnement de recherche

J'ai effectué mon stage dans le cadre du module d'Introduction à la Recherche en Laboratoire, proposée en 2ème année du cursus ingénieur ENSIMAG. Ce stage a eu lieu à l'INRIA Grenoble-Rhône-Alpges, situé à Montbonnot-Saint-Martin, au sein de l'équipe **CONVECS**, qui est une équipe de recherche commune à l'INRIA Grenoble-Rhone-Alpes et au Laboratoire d'Informatique de Grenoble (LIG).

L'équipe CONVECS (CONstruction of VErified Concurrent Systems) de l'INRIA Rhône-Alpes réalise des recherches sur la vérification et modélisation formelle de systèmes concurrents asynchrones, à base de langages formels décrivant le comportement ou les propriétés de tels systèmes, ou encore d'algorithmes et outils de vérification formelle. Au sein de cette équipe, j'ai travaillé avec **Gwen SALAÜN** (enseignant-chercheur) et **Lakhdar AKROUN** (chercheur post-doctoral) sur le sujet de la stabilité des systèmes communicants. Ce projet s'inscrit dans ces thématiques de recherche habituelles du laboratoire, et s'appuie sur un papier récemment publié par ses membres [9]. A partir de cela, la travail s'est organisé entre nous trois sur des aspects théoriques et concrets, et en lien avec nos expériences respectives : j'ai pu bénéficier de leur connaissance théorique du domaine le tout en appliquant mes compétences de programmation.

1.2 Contexte scientifique et technique

1.2.1 Systèmes communicants

De très nombreux systèmes et programmes informatiques répartis peuvent être modélisés et fonctionnent comme des systèmes communicants, c'est à dire par des messages qui transitent via des canaux de communication, en particulier par une succession d'envois et de réceptions de ces messages. On peut citer par exemple des architectures Web Client/Serveur : le client va envoyer un message au serveur, et en attendre sa réponse. Mais ceci est aussi applicable dans de nombreux cas concrets, liés à l'informatique où non, puisque dès qu'un système est composé de "briques" fonctionnelles (logicielles ou non), il s'agit d'une forme de système communicant, car ces briques ont des interactions entre elles qu'elles doivent gérer.

La formalisation de tels systèmes communicants se par des systèmes de transitions étiquetés (abrégé en LTS, pour Labeled Transition System). Ils représentent chacun une brique fonctionnelle du système, où les différents états du système servent de nœuds, et les transitions représentent les échanges de messages. Chaque LTS du système étudié est appelé un "pair" (parfois remplacé par son original anglais, peer).

Définition 1 (LTS). Un pair est un LTS $\mathcal{P} = (S, s^0, \Sigma, T)$ où S est un ensemble fini d'états, $s^0 \in S$ est l'état initial du système, $\Sigma = \Sigma^! \cup \Sigma^? \cup \{\tau\}$ est un alphabet fini partagé en un ensemble de messages d'émission, de réception et de l'action interne, et $T \subseteq S \times \Sigma \times S$ est une relation de transition.

Pour tout message, on utilise un symbole différent selon la nature du message : par convention, le nom du message est par exemple suivi d'un "!" si il s'agit d'un envoi, et d'un "?" s'il s'agit d'une réception.

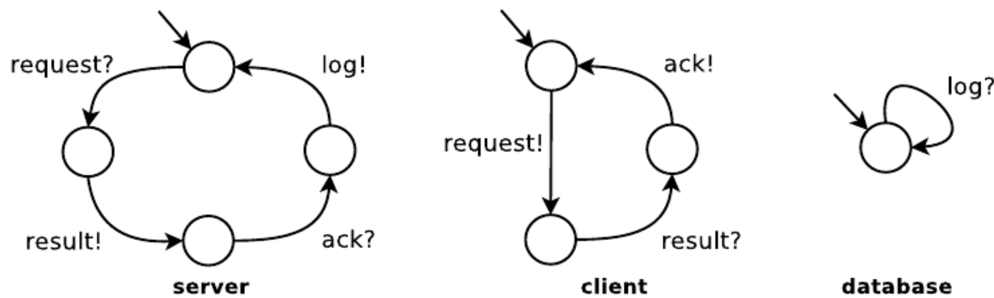

Figure 1 – Exemple de système modélisé par des LTS

Dans la suite, on admettra que ces pairs sont déterministes au niveau des messages observables, c'est-à-dire que si plusieurs transitions partent d'un état, et que tous les messages représentés sont bien observables, alors toutes ces transitions sont différentes l'une de l'autre.

1.2.2 Communication asynchrone et comportement d'un système

Les systèmes communicants peuvent fonctionner de façon synchrone (c'est le cas d'une communication en "temps réel", où quand l'émetteur envoie son message, le récepteur est forcément en état de pouvoir recevoir ce message) ou asynchrone (dans ce cas-là, à l'inverse, l'état n'importe pas : les messages n'ont alors pas de conditions précises sur l'état de l'émetteur ou du récepteur, dans le sens où le récepteur n'a pas besoin d'être en état de réception exactement au moment où l'émetteur envoie son message.).

Dans notre cas, tous les systèmes sont considérés asynchrones au départ. Les messages envoyés mais non reçus par le destinataire sont conservés dans des buffers de type FIFO : chaque peer est alors associé à un buffer infini (pour éviter des contraintes de modélisation). Chaque peer peut alors soit envoyer un message en queue du buffer vers un état accessible depuis l'état courant, ou lire un message présent en tête de son buffer, ou bien évoluer indépendemment des envois de messages par une action interne.

Toutefois, ces buffers non bornés nécessitent donc potentiellement un espace d'états infini pour représenter tous les états possibles du système. À cette contrainte technique s'ajoute un problème formel de plus grande échelle et important dans ce domaine de travail : le fait de savoir si un système se comporte comportement (pas d'interblocage, par exemple), et même d'analyser de façon général les propriétés d'un système communicant asynchrone, est indécidable [3].

Pour pallier à ce problème, la méthode généralement préconisée est d'utiliser des techniques de vérification formelles utilisées pour les systèmes finis, en cherchant des sous-classes de systèmes infinis analysables. Il devient alors possible d'analyser formellement le comportement précis d'un certain type de systèmes, ce qui a été fait par exemple pour pouvoir vérifier la **synchronisabilité** d'un système, c'est-à-dire s'il se comporte de la même façon pour toutes les bornes de buffers possibles, commençant à 1 [1] : on va vérifier si les action d'émissions générées par le pair se comportement de la même façon en communication synchrone et asynchrone. Le cas échéant, le traitement en est simple, puisque si le système est synchronisable, il est possible d'étudier son produit synchrone, ce qui a par exemple été fait récemment pour analyser des propriétés de systèmes communicants. [7]

Le produit synchrone d'un ensemble de pairs correspond donc à l'état du système où une communication a forcément lieu quand un pair peut envoyer un message, et l'autre est dans un état où il peut recevoir ce message. Ce produit est alors la composition des différents LTS du système en un seul automate, plus complexe, mais modélisant le même système en un seul LTS.

On peut également réaliser un produit similaire, mais modélisant les communications asynchrones possibles du système. Le LTS résultant est alors forcément plus complexes, car devant modéliser plus de combinaisons possibles, les émissions et réceptions d'un même message n'ayant plus de contrainte de simultanéité.

Définition 2 (Produit asynchrone). Soient un ensemble de pairs $\{\mathcal{P}_1, ..., \mathcal{P}_n\}$ avec $\mathcal{P}_i = (S_i, s_i^0, \Sigma_i, T_i)$, et Q_i son buffer associé. Alors le produit asynchrone est le LTS suivant : $LTS_a = (S_a, s_a^0, \Sigma_a, T_a)$, où :

- $S_a \subseteq S_1 \times Q_1 \times ... \times S_n \times Q_n$ où $\forall \in \{1, ..., n\}, Q_i \subseteq (\Sigma_i^?)^*$
- $s_a^0 \in S_a$ tel que$s_a^0 = (s_1^0, \epsilon, ..., s_n^0, \epsilon)$ (où ϵ représente un buffer vide)
- $\Sigma = \cup_i \Sigma_i$
- $T_a \subseteq S_a \times \Sigma_a \times \Sigma_a$, et pour $s = (s_1, Q_1, ..., s_n, Q_n) \in S_a$ et $s' = (s'_1, Q'_1, ..., s'_n, Q'_n) \in S_a$
  (envoi) $s \xrightarrow{m!} s' \in T_a$ si $\exists i, j \in \{1, ..., n\}$ où $i \neq j : m \in \Sigma_i^! \cap \Sigma_j^! =?$, alors :
  (1) $s_i \xrightarrow{m!} s'_i \in T_i$
  (2) $Q'_j = Q'_j m$
  (3) $\forall k \in \{1, ..., n\} : k \neq j \Rightarrow Q'_k = Q_k$, et
  (4) $\forall k \in \{1, ..., n\} : k \neq i \Rightarrow s'_k = s_k$
  (lecture) $s \xrightarrow{\tau} s' \in T_a$ si $\exists i \in \{1, ..., n\} : m \in \Sigma_i^?$, alors :
  (1) $s_i \xrightarrow{m?} s'_i \in T_i$
  (2) $mQ'_i = Q'_i$
  (3) $\forall k \in \{1, ..., n\} : k \neq i \Rightarrow Q'_k = Q_k$, et
  (4) $\forall k \in \{1, ..., n\} : k \neq i \Rightarrow s'_k = s_k$
  (action interne) $s \xrightarrow{\tau} s' \in T_a$ si $\exists i \in \{1, ..., n\}$, alors :

(1) $s_i \xrightarrow{\tau} s'_i \in T_i$
(2) $\forall k \in \{1, ..., n\} : Q'_k = Q'_k$, et
(3) $\forall k \in \{1, ..., n\} : k \neq i \Rightarrow s'_k = s_k$

Les deux figures suivantes (2 et 3et [5] illustrent ces deux types de produit. Sur ce type de LTS, des conditions (dires "gardes") sont utilisées, car ces exemples utilisent des LTS dites "machines à états étendues", où les transitions sont décrites par une condition et une action (à la manière d'un diagramme UML d'états-transitions), ce qui n'est pas le cas sur nos LTS. Le comportement et le fonctionnement des produits reste toutefois le même.

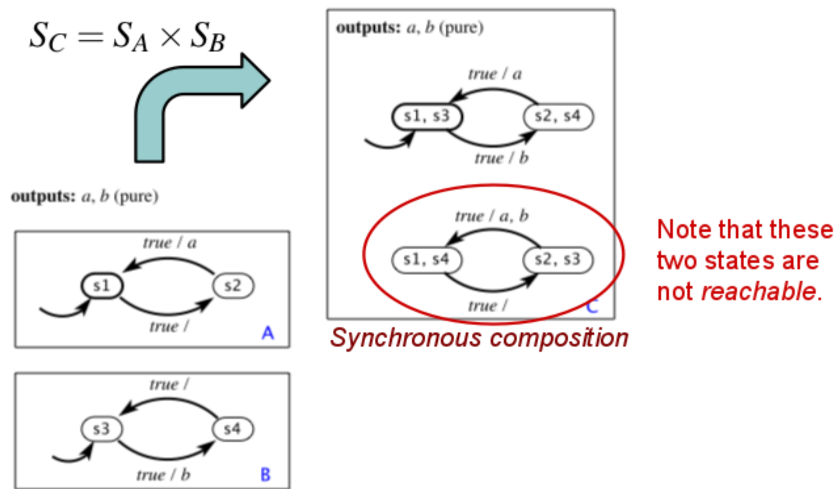

FIGURE 2 – Exemple de produit synchrone entre deux machines à états

Une propriété dite de synchronisabilité a pu être identifiée, correspondant à des systèmes ne créant pas d'espaces d'états infinis lors de communications asynchrones sans limite de buffer. Elle est validée quand les envois présents dans le système se comportement de la même façon si le système communique de façon synchrone ou asynchrone ; il faut alors réaliser des tests d'équivalence entre les deux compositions. Toutefois, cette propriété est assez forte, et peu de systèmes sont finalement synchronisables (par exemple, le système présenté en figure 1 ne l'est pas). Dans ce cas, il faut chercher d'autres sous-classes de systèmes qui seraient analysables, mais aucune n'approche n'existait. Ceci amène alors à la notion récente de **stabilité d'un système communicant**.

1.2.3 Résultats de départ du projet et état de l'art

Il est possible qu'un système communicant, composé d'un ensemble fini de pairs, et utilisant des communications asynchrones, se stabilise à partir d'une certaine borne de buffer spécifique k. Cette propriété, une fois montrée, est appelée "stabilité" du système communicant, qui est qualifié de "stable". Il s'agit d'un résultat récent de l'équipe CONVECS [9], qui a été la base de ce sujet d'IRL.

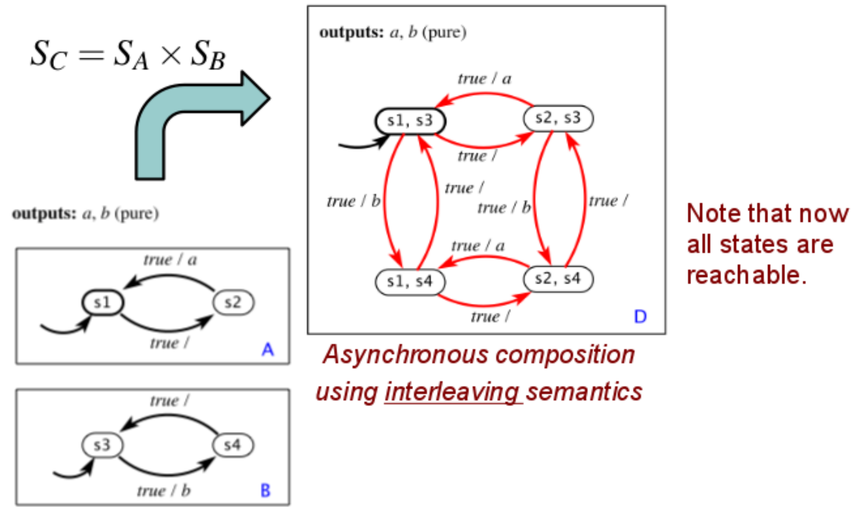

FIGURE 3 – Exemple de produit asynchrone entre deux m

La conséquence principale de ce résultat étant que si le système ne change pas de comportement à partir d'un certaine k, on peut analyser ses propriétés à ce k précis, par exemple pour la montrer l'absence d'interblocages, ou d'autres analyses structurelles : puisque le système est stable, chaque propriété de ce type qui sera montrée à la borne k sera valide pour toutes les bornes supérieures, et également lorsqu'on considérera des buffers considérés "infinis".

Définition 3 (Stabilité). Soit un ensemble de pairs $\{\mathcal{P}_1, ..., \mathcal{P}_n\}$. Le système correspondant à ces pairs est stable si et seulement si $\exists k$ tel que $LTS_a^k \equiv_{br} LTS_a^q, \forall q > k$.

Où "$\equiv_{br}$" correspond à une équivalence par branchement [10], c'est-à-dire à une relation R symétrique entre les deux LTS telle que les états initiaux sont liés par R, et que, avec r et s des états respectifs du premier et du second LTS, si $R(r,s)$ et $r \xrightarrow{\delta} r'$, alors soit $\delta = \tau$ et $R(r', s)$, soit il existe un chemin $s \xrightarrow{\tau^*} s_1 \xrightarrow{\delta} s'$ tel que $R(r,s)$ et $R(r', s')$.

Il a également été montré en [9] qu'il est indécidable en général de vérifier si un système asynchrone est stable, découlant du problème de décidabilité de comportement d'un système asynchrone ([3]), également qu'un système synchronisable est stable, et aussi que la borne k minimale d'un système stable est toujours calculable.

A partir de ceux travaux théoriques, le but est d'exploiter cette notion de stabilité, assez puissante, pour en améliorer la manipulation, et trouver des méthodologies simples permettant de trouver des systèmes stables. Notamment, améliorer les heuristiques existantes, et étudier théoriquement comme techniquement le fonctionnement de celles-ci.

2 Analyse de problèmes ouverts

Ce projet s'est effectué sur un sujet et des résultats précis et concrets, mais le spectre des champs de recherche possibles étant assez large, il a fallu effectuer une étude globale des aspects de travail envisagés à l'écriture du sujet, et d'améliorer l'existant ; une telle étude étant à la fois théorique et formelle, mais aussi technique, avec une analyse du code écrit et utilisé pour trouver le résultat de [9].

On a vu que la stabilité, en plus d'être une propriété plutôt forte (donc peu présente), est indécidable. Par ailleurs, elle est actuellement plutôt coûteuse à montrer : on cherche alors à trouver sur quels paramètres influer pour pouvoir répondre plus vite à cette question de stabilité d'un système. Différentes pistes ont été explorées dans ce but durant les semaines de travaux auxquels j'ai participé.

2.1 Analyse structurelle de systèmes

Certains systèmes communicants sont structurellement stables ou instables ; c'est-à-dire que l'on sait dire, d'après des caractéristiques structurelles qu'ils présentent, s'il existera ou non une borne de stabilité qu'on pourra trouver par vérification formelle ensuite. Ces systèmes sont le plus souvent très simples, mais ils permettent d'éliminer a priori, par un pré-traitement, toute une famille de systèmes qu'on aurait à traiter. Cette piste avait déjà été explorée en étudiant des systèmes dits "finis", c'est-à-dire dont l'interaction se fait sur des buffers non-bornés et où les produits asynchrones sont finis : ceci comprend les pairs sans cycles, ou les systèmes avec un seul pair cyclique. [6]

Pour mettre en place ce pré-traitement, il faut trouver quoi détecter dans les pairs de chaque système, et définir des critères structurels d'élimination qu'il faudrait ensuite mettre en place dans l'existant, et valider expérimentalement.

2.2 Calcul des bornes de buffers

Les bornes de buffers testées pour la stabilité d'un système par le code en place ne sont pas choisies aléatoirement ou linéairement. En général, elles consistent en des heuristiques pour établir une borne de départ puis en l'utilisation d'algorithmes de recherche pour trouver la borne à partir de laquelle il y a stabilité.

Des exemples déjà utilisés consistent en par exemple prendre la plus longue séquence d'émissions comme borne de départ, ou encore calculer le nombre maximum d'émissions destinées au même pair, et prendre ce nombre comme valeur de départ. Les algorithmes de recherche utilisés sont en général une recherche dichotomique, ou une simple incrémentation/décrémentation de valeur. Ces méthodes ont des efficacités variables selon les systèmes traités, mais il arrive que certains durent plusieurs dizaines de minutes avant de donner une borne, voire tournent finalement plus d'une heure sans résultat [9]. Dans ces cas-là, on ne peut rien conclure sur la stabilité du système étudié.

Une perspective serait alors un moyen d'optimiser ce calcul de bornes, en trouvant notamment des heuristiques de calcul plus efficaces ou optimales. Toutefois, on peut supposer sur

des systèmes complexes, les heuristiques seront loin d'être évidentes, et qu'au contraire sur des systèmes plus simples, les heuristiques et algorithmes existants semblent plutôt efficaces.

2.3 Performances

Le calcul de stabilité est une opération très longue et coûteuse. Ceci car il est composé de 2 phases, réalisées par des outils externes maintenus par l'équipe du laboratoire (la boite à outils CADP [4]), qui réalise des opérations formelles coûteuses. Il s'agit du calcul des produits d'automates itératifs, réalisés en boucle et sans optimisations, ainsi que des tests d'équivalence entre ces automates complexes (contenant par exemple plusieurs centaines d'états et de transitions) [2].

Il est difficile d'influer sur le calcul, car les opérations internes au programme en place sont peu coûteuses, et les outils externes de CADP n'ont rien à voir avec le script du projet en lui-même, et il est donc compliqué de toucher précisément aux calculs faits par les outils. Il faudrait s'immerger dans le code propre des outils et recoder ces opérations complexes, ce qui est impossible à réécrire de zéro et plus simplement dans le cadre de l'IRL.

On peut par contre envisager d'optimiser les appels réalisés à CADP, notamment en stockant certaines valeurs (les produits d'automates, par exemple) qui n'auraient pas besoin d'être recalculées.

2.4 Uniformité des bornes

Le système en place calcule une borne minimale de stabilité pour le système complet ; c'est-à-dire que la même borne sera appliquée à chaque pair du système. Mais on peut se demander pourquoi chaque pair aurait-il besoin de la même taille de buffer : il est possible que moins de place de stockage soit nécessaire, et qu'une optimisation soit possible à ce niveau, puisqu'avec une borne exacte calculée pour chaque pair, le calcul itératif sera plus rapide, et les automates calculés seront plus petits.

Toutefois, cela ajouterait potentiellement un sur-coût au calcul des bornes, et rendrait finalement le calcul plus long de façon globale ; l'idée est bonne en théorie, mais pourrait avoir des répercussions sur le coût du calcul. Cette piste reste donc à l'étude.

3 Contributions théoriques et formelles

Après études des différents aspects des problèmes évoqués ci-dessus, l'orientation du travail a été décidée, allant vers la réflexion et l'implémentation d'un pré-traitement pour la détection de stabilité (ou de non-stabilité) de systèmes, et permettre ainsi d'accélérer le temps de traitement en utilisant des algorithmes d'analyse structurelle, détectant des cas à ne pas traiter formellement par CADP.

Cette approche présente moins de blocages et de besoins théoriques, qui auraient été potentiellement plus longs à comprendre pour pouvoir les manipuler ensuite, et reste tout aussi utile à l'analyse de systèmes existants, notamment pour des travaux futurs.

3.1 Pré-traitement de systèmes

Le principe du pré-traitement à mettre en place est simple. Il s'agit d'implémenter une série de fonctions par lesquelles va passer le programme avant d'exécuter les calculs de bornes et les tests d'équivalence, opérations coûteuses comme vu auparavant, afin de pouvoir interpréter et analyser ces systèmes, et constater s'il est possible ou non de dire quelque chose sur la stabilité d'un système avant de passer par des vérifications formelles plus poussées. L'objectif concret de ce projet est de trouver comment réaliser ce pré-traitement efficacement, et d'en réaliser une implémentation propre dans le code existant.

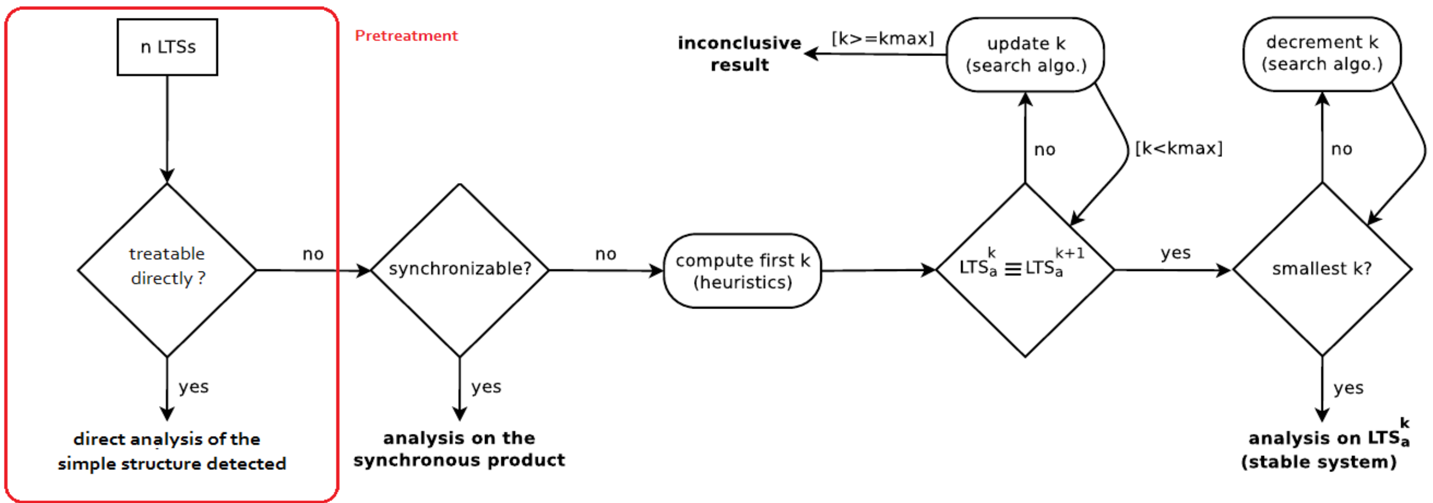

Figure 4 – Méthodologie générale de calcul de stabilité

Sur cette figure (tirée de [9]), le rôle du pré-traitement se situe entre avant la vérification de synchronisabilité, sur la gauche de la figure. Si le pré-traitement arrive à détecter une structure connue, qu'il est possible d'interpréter, le programme terminera, et "sortira" de ce schéma, ce qui permet d'éviter une bonne partie des opérations de la méthode classique.

3.2 Dépendances de cycles

Nous avons vu dans la section 2.1 qu'un des buts principaux était de discerner des catégories de systèmes sur lesquelles il est possible de dire quelque chose sur la stabilité. Nous avons pu étudier durant plusieurs semaines quelles catégories il était possible de définir, en partant de plusieurs formes simples, qui sont devenues progressivement plus poussées, des LTS qu'il est possible de traiter, d'après les éléments théoriques connus. Voici les structures qui ont eu le temps d'être explorées :

- Systèmes séquentiels : un système contenant uniquement des pairs sous forme de séquences (sans aucun cycle) est forcément stable. En effet, les pairs ne font que rester en place en attendant un message, soit continuent jusqu'à une fin précise. Dans tous les cas, il est défini que le système sera à l'arrêt à un temps donné.

- Systèmes à cycle unique : lorsque les pairs ne sont composés que d'un seul cycle (et donc le reste correspondant à des séquences), deux cas sont possibles.
  - Cycle d'émissions : si le cycle est un cycle composé exclusivement d'émissions, le système n'est pas stable, car si le système peut envoyer des messages en boucle, il est possible qu'il ne s'arrête jamais, et donc les produits d'automates à chaque borne de buffer possible seront toujours différents.
  - Autre type de cycle : dans tous les autres cas, le système est stable, puisque le cycle peut s'apparenter à une séquence quelconque mise en boucle ; tant que la boucle contient des réceptions, le système peut s'arrêter, et on peut donc le voir se stabiliser.

- Systèmes à cycle d'émission : on peut généraliser le résultat vu plus haut sur les cycles d'émission. Si un pair contient un cycle d'émission en général, peu importe les autres cycles et l'état des autres pair, le système n'est pas stable.

- Systèmes à un seul peer cyclique : Si un seul pair du système contient des cycles (hors cycle d'émission), alors le système est stable. En effet, si les autres pairs sont séquentiels, on peut toujours avancer dans l'exécution du système, et les autres pairs se "termineront". Le pair cyclique sera alors à l'arrêt ou déjà terminé, et le système est donc stable.

- Systèmes à cycles indépendants : lorsque les pairs contiennent plusieurs cycles, et qu'ils sont indépendants (c'est-à-dire qu'ils n'ont aucun message en commun, en émission comme en réception), alors le système est stable. En effet, puisqu'il n'y a aucun interdépendance, ils se comportement comme si chaque cycle était dans un pair distinct, par exemple.

- Systèmes à cycles dépendants (et cycle au démarrage) : lorsque les cycles des pairs contient les mêmes données de message, il faut faire des études plus précises, car leur comportement dépend de l'interdépendance de ces cycles. Dans cette étude, on étudiera uniquement des pairs qui **démarrent par un cycle**. En effet, dans le cas contraire, l'état des buffers dépend de la séquence qui existerait avant le premier cycle, et donc l'étude serait beaucoup plus complexe, notamment si la séquence est longue. Dans la lignée des cas précédents, tous les cycles présentés ici ne sont pas des cycles d'émission, mais tous les autres types de cycles possibles.
  - Avec même séquences de messages : si les cycles ont exactement le même ordre de messages (mais sont par exemple inversés en termes d'émission/réception), le système est stable. En effet, dans ce cas-là, le système avance de façon synchrone, et on se retrouve alors avec les mêmes propriétés que la synchronisabilité décrite plus haut.
  - Cycles simples séparés par des séquences : le système est stable s'il se finit par une séquence (système fini tel que décrit plus haut), sinon il se comporte de la même façon que les cycles suivants. L'étude théorie s'est arrêtée à ce niveau, les interprétations devenant moins triviales à donner et les méthodologies plus compliquées à abstraire.

3.3 Perspectives

De nombreux exemples ont été étudiés pour définir les catégories à "éliminer" par prétraitement. Une fois les structures simples éliminées des systèmes à traiter, un constat est

net : la complexité des systèmes restants augmente rapidement. Dans la majorité des cas, ces LTS présentent en fait des structures complexes, comprenant des cycles imbriqués, interdépendants, et sans propriété particulière (comprenant un mélange d'émissions et de réceptions, par exemple).

La méthodologie utilisée est alors moins efficace, car elle mènerait rapidement à devoir développer des algorithmes et créer des catégories seulement pour un faible nombre de systèmes, mais en grande quantité. En effet, il n'est pas simple de rendre générique des algorithmes poussés sur la détection de certains types de cycles, et le travail serait alors plus complexe, moins intuitif, et moins pertinent que celui réalisé pour définir les catégories définies plus tôt.

4 Réalisations pratiques et résultats

4.1 Code existant

Un autre pan principal du projet a été de reprendre comme base un code déjà existant. Il consistait en un fichier unique, conséquent, écrit en Python, et interagissant avec la "boite à outils" externe CADP évoquée plus tôt, développée et maintenue par l'INRIA depuis plusieurs années [4], et qui permet d'effectuer des opérations sur les systèmes communicants décrits par un format précis (dans un langage de haut niveau, simple à écrire, qui est ensuite parsé par ces outils).

Le code étant conséquent, le but était de réaliser un nettoyage et un refactoring du code, pour améliorer sa lisibilité et son organisation du code. La tâche n'étant pas d'une grande difficulté en termes de programmation (peu de méthodes avaient à être modifiées, notamment sémantiquement, et le code écrit en Python objet est clairement lisible), la complexité du travail résidait surtout dans le fait devoir parcourir tout le code à de nombreuses reprises pour en tirer le sens global et tous les contextes d'utilisation possibles, afin d'en déduire la chaîne d'exécution général. Ainsi, à partir d'un simple fichier "async.py", la structure a été modifié, comme montré sur cette figure :

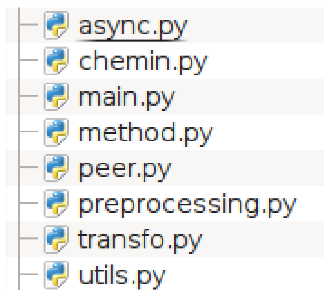

FIGURE 5 – Organisation du code Python

Chaque fichier correspondant à une classe Python, contenant respectivement des méthodes liées à son cadre d'utilisation.

Un des problèmes potentiels présents dans ce travail est que certaines fonctions étaient codées sans être utilisées, simplement par anticipation théorique des étapes qui allaient suivre

dans les travaux théoriques en cours. Il n'était alors pas forcément aisé de savoir si ces parties de code allait être à re-coder ou non, car une simple description de la fonction et de ce qu'elle fait n'indique pas toujours son utilisation exacte.

4.2 Algorithmes et tests

En clair, ce travail d'IRL s'est axé autour de réflexions théoriques combinées à des méthodologies pour implémenter concrètement ces travaux formels. Une grosse part consistait alors en la réflexion concernant des algorithmes de détections de structures sur les systèmes asynchrones, puis leur implémentation et leur test en Python dans les scripts du projet. Certains algorithmes ont pu être implémentés, fonctionnant de manière similaire, mais s'appliquant à chaque pour des LTS différents, et ayant pour chacune de ces catégories des spécificités d'analyse. De manière analogue aux systèmes décrits en 3.2, on définit des heuristiques pour chaque structure :

- **Détection de séquences** : Pour détecter des séquences dans LTS, on parcourt chaque pair, et dès qu'une transition amène à un état déjà parcouru (et donc qu'il y a présence d'un cycle), on sort de la fonction : le système n'est pas séquentiel. Dans le cas contraire, il est stable car fini (ses produits asynchrones le sont).

- **Cycles uniques** : Il s'agit de détecter des pairs contenant un unique cycle. On réalise le même parcours des transitions, et on se garde cette fois une tolérance pour un cycle ; au premier cycle détecté, on reprend une détection de séquence comme auparavant, et on sort si un nouveau cycle est détecté. Si le cycle détecté est entièrement composé d'émissions, alors le système est instable. Dans le cas contraire, avec un cycle unique quelconque, le système est stable.

- **Détection de cycles indépendants** : On réalise un algorithme de détection de cycles, fonctionnant comme dans un graphe orienté [8], en récupérant tous les messages concernés par ces cycles. A chaque nouveau cycle, on vérifie si le cycle courant possède des étiquettes communes avec l'ensemble stocké ; si oui, on sort : il y a des cycles interdépendants. Dans le cas contraire, tous les cycles sont indépendants, et le système se comporte comme une séquence et est stable.

- **Détection de cycles dépendants** : la détection de cycles interdépendants a été effectuée en utilisant des cas plus simples : avec des systèmes à 2 pairs maximum, ainsi qu'avec un cycle en début de graphe. Pour détecter ce cycle initial, on lance une détection de cycle classique, en attendant de revenir à l'état initial ; si ce n'est pas le cas, il n'y a pas de cycle en début de graphe (ou il y en a un, avec d'autres cycles imbriqués ; ces cas-là étant trop complexes pour les traiter dans le cadre de l'étude, ils sont ignorés).
  - cycles à même séquence de messages : toujours uniquement sur les systèmes à 1 (qui est ignoré car traité plus tôt) ou 2 pairs, on procède en parcourant les cycles initiaux des 2 pairs, et en comparant les séquences de messages détectées. Si elles sont identiques, on quitte la fonction : le système est stable.
  - cycles à alphabet complémentaires : on procède de la même façon, on vérifiant cette fois que, si la séquence n'est pas la même, les messages se complément proprement

c'est-à-dire qu'il n'y a pas plus d'émissions que de réceptions pour un même message une fois le parcours terminé.

- Pour les derniers systèmes (cycles séparés par des séquences (avec fin par un cycle ou un séquence), ...) ainsi que les études suivantes, des ébauches d'algorithmes ont été réalisées mais non implémentées, et sont donc restées à l'état d'hypothèses ; elles concernent notamment la détection de cycles imbriqués, ou la gestion de cycles dans des systèmes commençant par une séquence.

Il est à noter que même si ces algorithmes sont très similaires, ils ont été implémentés dans des fonctions séparées, pour faciliter à la fois la mise en place et la validation des fonctions. Pour des développements futurs, il sera sans doute possible de factoriser des parties de ces algorithmes, mais toutefois ce dédoublement de morceaux de fonctions n'a que peu d'implications sur l'exécution, car le traitement se fait de toute manière rapidement (surtout comparé aux tests formels éventuels après ce pré-traitement).

Pour valider ces algorithmes, plusieurs centaines de LTS avaient été décrits dans le format de haut niveau cité plus haut, de structures plus ou moins complexes, et modélisant parfois des systèmes réels : les tests ont été effectués sur cette base d'automates. Par ailleurs, j'ai pu développer moi-même mes propres exemples en me basant sur ceux qui étaient déjà présents, après une initiation à CADP : le schéma est simple, comprenant un fichier par "pair" du système, et chaque ligne de code (excepté la première, qui indique le nombre total de sommets et de transitions) indique une transition, à la fois sa source, sa destination, et le message qu'elle porte.

Pour tester plus rapidement les fonctions de traitement, j'ai notamment écrit un script en Bash, testant le pré-traitement sur tous les exemples disponibles, et classant les résultats pour associer une quantité à chaque type de système détecté. Il s'agit d'une manière simple et évolutive d'évaluer mes algorithmes, avec possibilité de vérifier à la main si les données sont correctes après coup, qui m'a été confirmée comme une méthodologie à suivre en général pour tester ce genre de fonctions.

4.3 Résultats et observations

En utilisant tous les algorithmes développés pendant le projet sur la base de systèmes disponibles, on obtient les résultats suivants :

- Sur 307 systèmes disponibles, 163 ont pu être interprétés par le pré-traitement, soit plus de la moitié dont on a pu éviter un traitement coûteux ;

- Parmi ces 163 systèmes :
  - 30 étaient séquentiels, et par conséquent stables ;
  - 83 comportaient un seul cycle, parmi lesquels 23 étaient stables (aucun cycle d'émission présent), les 60 restant étant instables car ce cycle était un cycle d'émission ;
  - 12 comportaient des cycles indépendants, et étaient par conséquent stables
  - 7 avaient des cycles dépendants utilisant la même séquence de message, avec un cycle au démarrage des pairs.

– Enfin, 31 avaient des cycles dépendants et un cycle au démarrage du système, avec des cycles utilisant des alphabets complémentaires entre chaque pair.

Ces résultats sont plutôt encourageants, car beaucoup de ces systèmes modélisaient des situations réelles, et donc donnant des systèmes qu'il seraient possibles d'étudier concrètement.

Par ailleurs, un avantage non négligeable est le gain de performances. Outre le fait d'enlever des traitements formels très coûteux (exponentiel et polynomial), le pré-traitement est lui de coût linéaire, réalisant des simples boucles sur chaque transition (ou chaque état) de chaque pair (en général, il y a entre 1 et 4 pairs par système), de façon récursive, et assez régulièrement en s'arrêtant avant la fin de l'algorithme. L'algorithme est alors parfois polynomial (en $O(x^3)$), mais les valeurs étant finies et faibles, le temps effectif de calcul est en général bien plus bas, proche d'une complexité quadratique, voire linéaire.

5 Conclusion

5.1 En résumé

Le travail ayant surtout une portée théorique, la majeure partie du travail était tournée vers de l'analyse formelle, mais a permis de clarifier les perspectives éventuelles concernant à la fois le comportement de systèmes asynchrones, et l'analyse structurelle de ceux-ci ; mais des avancées techniques ont pu être faites également.

Notamment, en étudiant la proportion de systèmes éliminés sur cette panoplie de systèmes classiques et réels, il a été possible de remarquer que beaucoup de systèmes sont en réalité relativement complexes, même lorsqu'ils modélisent des systèmes aux interactions simples. Ceux-ci sont alors difficiles à classer structurellement par des algorithmes simples.

Par conséquent, il devient difficile de jouer sur l'efficacité de la vérification de stabilité si même un pré-traitement n'arrive pas à éliminer la majorité des cas. Toutefois, il permet d'éviter une vérification couteuse en temps dans une partie non-négligeable des systèmes existants, qui sont certes les plus simples, mais qui sont aussi visibles dans le monde réel.

5.2 Bilan scientifique et personnel

L'apport scientifique de ce projet s'est fait à la fois sur la façon de réfléchir structurellement sur les systèmes communicants asynchrones, et aussi de manière plus technique sur l'apport d'une base conséquente de pré-traitement pour catégoriser ces systèmes. J'ai pu ainsi participer aux travaux exploitants cette notion intéressante de stabilité, pour aider à mieux la comprendre et déceler sa présence, en tout cas quels types de systèmes se prêtent à l'utilisation de cette notion de stabilité.

A titre personnel, ce projet a été très instructif, car il m'a permis de m'immerger dans un monde bien différent de celui de l'ENSIMAG, avec une équipe travaillant à temps plein sur un sujet fixe, mais aussi dans le monde de la recherche, que ce soit ces aspects "administratifs" (en assistant par exemple à des répétitions de membres de l'équipe pour des conférences), pour des rédactions de travaux, ou l'exploitation d'ancien résultats, mais aussi dans la méthodologie, et la façon de travailler à plusieurs même quand le sujet précis de recherche diffère selon les personnes.

5.3 Perspectives et travaux futurs

Après cette étude, il reste encore de nombreuses avancées à réaliser pour étayer cette notion de stabilité, qui porte un potentiel important pour l'étude de systèmes concurrents. On peut imaginer des études théoriques plus poussées au niveau des heuristiques, pour trouver des catégories plus précises de systèmes à éliminer à l'avance, notamment en utilisant des algorithmes plus efficaces, et plus génériques pour analyser les LTS.

Par ailleurs, le système en place est parfois imparfait, notamment dans la complexité du calcul de vérification de stabilité : il y a sans doute des améliorations de forme et de fond à

réaliser dans notamment les produits d'automates et les tests d'équivalence, par exemple en économisant des calculs par stockage d'anciens résultats, en utilisant des méthodes de programmation dynamique.

Remerciements

Je tiens à remercier toute l'équipe CONVECS de l'INRIA Grenoble-Rhône-Alpes, qui m'a accueilli pendant 3 mois avec gentillesse et sans aucun souci pendant toute la durée de l'IRL. Particulièrement, je remercie Gwen SALAÜN et Lakhdar AKROUN pour leur encadrement théorique et technique, ainsi que leurs conseils et informations pertinentes, que ce soit sur la thématique du projet ou les méthodologies de recherche en général, et je remercie également Frédéric LANG, Radu MATEESCU, Hugues EVRARD, Eric LEO, Hubert GARAVEL et Fatma JEBALI pour tout ce qu'ils ont pu apporter à ma connaissance du monde de la recherche, et tout ce dont on a pu discuter pendant ces 12 vendredis passés au laboratoire, peu importe le sujet. Ils ont en fait en sorte que cette expérience soit bonne pour moi, ce qui a été le cas.

Par ailleurs, je remercie l'ENSIMAG d'avoir mis en place un tel module, car je trouve le monde de la recherche finalement peu mis en valeur dans le cursus de l'école, alors qu'il reste un débouché et un secteur intéressant, et que nombre de membres du personnel en savent quelque chose.

Références

[1] Basu, S., Bultan, T., and Ouederni, M. Deciding choreography realizability. *SIGPLAN Not. 47*, 1 (Jan. 2012), 191–202.

[2] Bergamini, D., Descoubes, N., Joubert, C., and Mateescu, R. Bisimulator : A modular tool for on-the-fly equivalence checking. In *Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2005, pp. 581–585.

[3] Brand, D., and Zafiropulo, P. On communicating finite-state machines. *J. ACM 30*, 2 (Apr. 1983), 323–342.

[4] Garavel, H., Lang, F., Mateescu, R., and Serwe, W. CADP 2011 : a toolbox for the construction and analysis of distributed processes. *International Journal on Software Tools for Technology Transfer 15*, 2 (2013), 89–107.

[5] Lee, E. A., and Sangiovanni Vincentelli, A. Introduction to embedded systems, 2014. EECS 149, UC Berkeley.

[6] Leue, S., Ştefănescu, A., and Wei, W. *Dependency analysis for control flow cycles in reactive communicating processes*. Springer, 2008.

[7] Ouederni, M., Salaün, G., and Bultan, T. Compatibility checking for asynchronously communicating software. In *Formal Aspects of Component Software*. Springer, 2014, pp. 310–328.

[8] Panhaleux, A. Recherche de cycles dans les graphes, 2007. ENS Lyon.

[9] Salaün, G., and Ye, L. Stability of Asynchronously Communicating Systems. Research Report RR-8561, July 2014.

[10] van Glabbeek, R. J., and Weijland, W. P. Branching time and abstraction in bisimulation semantics. *J. ACM 43*, 3 (May 1996), 555–600.