Parallelism and modular proof in differential dynamic logic

Simon Lunel

▶ To cite this version:

Simon Lunel. Parallelism and modular proof in differential dynamic logic. Artificial Intelligence [cs.AI]. Université de Rennes, 2019. English. NNT : 2019REN1S005 . tel-02102687

HAL Id: tel-02102687
https://theses.hal.science/tel-02102687

Submitted on 17 Apr 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

UNIVERSITE BRETAGNE LOIRE / MATHSTIC

UNIVERSITÉ DE RENNES 1

THÈSE DE DOCTORAT DE

L'UNIVERSITE DE RENNES 1

L'UNIVERSITE DE RENNES 1
COMUE UNIVERSITE BRETAGNE LOIRE

Ecole Doctorale N°601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : Informatique

Par

Simon Lunel

« Parallelism and modular proof in Differential Dynamic Logic »

Soutenance de la thèse prévue à Rennes, le 28 janvier 2019
IRISA

Rapporteurs avant soutenance :

AIT AMEUR Yamine, Professeur, INPT-ENSEEIHT, Toulouse, France
ZHAN Naijun, Professeur, Chinese Academy of Science, Beijing, China

Composition du jury :

Président :	BLAZY Sandrine, Professeure, ISTIC, Université de Rennes 1, Rennes, France
Examinateurs :	AIT AMEUR Yamine, Professeur, HDR, INPT-ENSEEIHT, Toulouse, France
	ZHAN Naijun, Professeur, Chinese Academy of Science, Beijing, China
	GAO Sicun, Assistant Professor, University of California, San Diego, USA
	BOYER Benoit, Chercheur, Mitsubishi Electric R&D Centre Europe, Rennes, France
Invité(s) :	GHORBAL Khalil, Chercheur, CNRS, IRISA, Rennes, France
Dir. de thèse :	TALPIN Jean-Pierre, H, Chercheur, HDR, INRIA, Rennes, France

Remerciements

Je tiens à remercier Sandrine BLAZY, Professeure de l'Université de Rennes 1, d'avoir accepté de présider mon jury. Je remercie également Yamine AIT AMEUR, Directeur de Recherche à l'INPT-ENSEEIHT, et Naijun Zhan, Professeur au Chinese Academy of Science, d'avoir bien voulu rapporter cette thèse et pour leur relecture minutieuse. Khalil GHORBAL, Chargé de recherche à l'INRIA, et Sicun GAO, Assistant Professor de l'Université de San Diego, ont accepté de faire partie de mon jury de thèse et je les en remercie.

Je tiens à remercier tout particulièrement mon directeur de thèse, Jean-Pierre TALPIN, Directeur de recherche à l'INRIA, pour m'avoir guidé tout au long de ces trois années de doctorat et sans qui je n'aurais pu accomplir ce travail. Ses retours ont toujours été pertinents et ont abondamment nourri ma réflexion. Je remercie aussi Benoit BOYER, Ingénieur de Recherche à Mitsubishi Electric qui m'a encadré sur toute la durée de mes travaux et qui a toujours été présent pour répondre à mes nombreuses questions. Son enthousiasme m'a aidé à continuer à de nombreuses reprises. Mes remerciements vont aussi à David MENTRE pour les nombreux échanges constructifs que l'on a eu dans les locaux de Mitsubishi Electric.

Je remercie aussi Denis COUSINEAU et Ocan SANKUR pour avoir été membre de mon comité de suivi de thèse ainsi que pour leur nombreux conseils et encouragements lors de ces discussions.

Ce manuscrit n'aurait pas été possible sans les nombreuses interactions et échanges que j'ai eu avec d'autres chercheurs, notamment André PLATZER, Stefan MITSCH, Andreas MULLER, Rajesh GUPTA, Shuling WANG. La liste n'est probablement pas exhaustive, et je tiens à m'excuser d'avance pour tout ceux que j'oublié. Je tiens aussi à remercier tous mes collègues de l'équipe TEA pour les échanges toujours intéressants et pour un soutien bien necessaire parfois : Loic BESNARD, Thierry GAUTIER, Vania JOLOBOFF, Hai Nam TRAN, Alexandre HONORAT, Jean-Joseph MARTY, Liangcong ZHANG et Lucas FRANCESCHINO. Merci à Stephanie LEMAILE et Armelle MOZZICONACCI pour leur précieuse aide dans les dédales administratifs de l'INRIA. Enfin, je remercie Mitsubishi Electric et l'INRIA pour avoir respectivement financé et soutenu matériellement ma thèse.

Je tiens à remercier mes parents pour leur soutien durant ma thèse ainsi que pour toute leur implication les vingts-trois années précédentes. Cette thèse n'aurait jamais vu le jour sans eux. Je remercie aussi mes amis à qui j'ai donné des nouvelles de manière erratique et qui ne m'en tiennent pas rigueur. Enfin, je remercie Lucie qui a réussi à me supporter durant ces trois années et m'a toujours soutenu.

Résumé en français

Les systèmes cyber-physiques mélangent des comportements physiques continus, tel la vitesse d'un véhicule, et des comportement discrets, tel que le régulateur de vitesse d'un véhicule. Ils sont désormais omniprésents dans notre société. Un grand nombre de ces systèmes sont dits critiques, *i.e.* une mauvaise conception entrainant un comportement non prévu, un *bug*, peut mettre en danger des êtres humains. Ainsi, un régulateur de vitesse qui autoriserait un véhicule à dépasser les limitations de vitesse met en danger les passagers du véhicule.

Il est nécessaire de développer des méthodes pour garantir le bon fonctionnement de tels systèmes. La méthode la plus répandue est le test. On va tester de manière approfondie si une voiture se comporte comme attendue ou si le programme réagit comme souhaité pour un grand nombre de situations possibles. Mais si cette approche permet de mettre en évidence des problèmes, elle n'est pas en mesure de garantir l'absence de problèmes. En effet, si on teste la voiture pendant un millier d'heures, il est possible qu'un problème arrive uniquement à partir de la mille-et-unième heure de fonctionnement. Une autre approche a été développé ces dernières décennies en informatique et qui permet de garantir l'absence problèmes : les méthodes formelles.

Les méthodes formelles regroupent des procédés mathématiques pour garantir qu'un programme se comporte comme attendu, par exemple que le régulateur de vitesse n'autorise pas de dépasser la vitesse maximale autorisée. Mais les procédés développés l'ont principalement été pour des programmes, *i.e.* que l'on ne considère pas la partie continue du système comme l'évolution de la vitesse, et on peuvent donc garantir le bon fonctionnement du système physique. Il est donc souhaitable de développer des méthodes formelles spécifiques pour garantir le bon fonctionnement de systèmes cyber-physiques.

Une telle tâche rencontre néanmoins un problème de taille qui est de réussir à conjuguer deux formalismes de natures différentes. En effet, les systèmes physiques sont représentés par des équations différentielles à valeur dans l'ensemble des nombre réels $\mathbb{R}$. Les programmes sont eux modélisés comme étant à valeur dans les entiers naturels $\mathbb{N}$. Cela est particulièrement problématique pour la représentation du temps. Quand un programme s'exécute, on ne s'intéresse pas au temps qui peut s'écouler entre deux instructions, la représentation du temps est *discrète*. Mais l'évolution d'un phénomène physique est *continue* : il n'y a pas de pause.

André Platzer a proposé en 2007 un nouveau formalisme, la logique dynamique différentielle, réunissant ces deux aspects. Ce formalisme permet de modéliser formellement, *i.e.* avec une sémantique définie mathématiquement, des systèmes exhibant à la fois des comportements continus et discrets, ainsi que leurs interactions possibles. De plus, André Platzer a défini un système de preuve permettant de raisonner sur de tel systèmes et donc de garantir leur bon fonctionnement. Il devient alors possible de modéliser formellement un régulateur de vitesse avec l'évolution réelle de la vitesse et de démontrer qu'il est impossible que le véhicule dépasse les vitesses maximales autorisées, cela pour un intervalle de temps arbitrairement long et non pour les mille premières heures.

Malheureusement, la mise en œuvre de cette méthode requiert une connaissance approfondie des systèmes étudiés ainsi que des mécanismes de preuves formelles associées. En effet, le système est représenté dans sa globalité, en un bloc monolithique, et la complexité de la preuve augmente en fonction de la taille du système. Ainsi, des systèmes relativement simples du point de vue d'un ingénieur dans l'industrie nécessitent un effort important de preuve qui ne peut être supporté par un ordinateur. Il nous faut donc développer des méthodes pour passer à l'échelle et augmenter la complexité des systèmes étudiés.

La complexité d'un système cyber-physique revêt deux aspects à notre connaissance. Le

premier aspect est la complexité que nous appelons *mathématique*. Il s'agit de la complexité inhérente à un phénomène physique, par exemple le déplacement d'un satellite où l'on est obligé de prendre en compte la théorie de la relativité. Le deuxième aspect est la complexité que nous appelons *structurelle*. Elle provient de la répétition de composants élémentaires, par exemple une usine de traitement des eaux où plusieurs cuves sont connectées ensemble. Chaque cuve prise individuellement est simple à comprendre, modéliser, et de là, à prouver son bon fonctionnement. Mais il est difficile de considérer l'usine en entier car la répétition rend la modélisation plus complexe ainsi que la preuve. Ces systèmes sont très courants dans les milieux industriels.

C'est ce dernier aspect que nous avons voulu traiter dans ce mémoire. Notre problématique est comment modéliser efficacement des systèmes cyber-physiques dont la complexité réside dans une répétition de morceaux élémentaires. Et une fois que l'on a obtenu une modélisation, comment garantir le bon fonctionnement de tels systèmes.

L'approche classique où l'on modélise le système d'un seul tenant et essaye ensuite de montrer son bon fonctionnement n'est pas applicable. La modélisation serait laborieuse et sujette à erreurs dues à la taille du modèle. La preuve du modèle résultant ne pourra pas être prise complètement en charge par un ordinateur car la puissance de calcul nécessaire est beaucoup trop élevée. Un expert en démonstration de système cyber-physique sera donc nécessaire.

Notre approche consiste à modéliser le système de manière compositionnelle. Plutôt que de vouloir le modéliser d'un seul tenant, il faut le modéliser morceaux par morceaux, appelés *composants*. Chaque composant correspond à une brique élémentaire du système et il est donc possible de le modéliser facilement. On obtient le système complet en assemblant les composants ensembles. Ainsi l'usine de traitement des eaux est obtenue en assemblant les cuves ensembles. L'intérêt de cette méthode est qu'elle correspond à l'approche des ingénieurs dans l'industrie : considérer des éléments séparés que l'on compose ensuite.

Mais cette approche seule ne résout pas le problème de la preuve de bon fonctionnement du système. Il faut aussi rendre la preuve *compositionnelle*. Pour cela, on associe à chaque composant des propriétés et on prouve qu'elles sont respectées. Cette preuve peut être effectué par un expert, mais aussi par un ordinateur si les composants sont de tailles raisonnables. Il faut ensuite nous assurer que lors de l'assemblage des composants, les propriétés continuent à être respectées. Ainsi, la charge de la preuve est reportée sur les composants élémentaires, l'assurance du respect des propriétés désirées est conservée lors des étapes de composition. On peut alors obtenir une preuve du bon fonctionnement de systèmes industriels avec un coût de preuve réduit.

Notre contribution majeure est de proposer une telle approche compositionnelle à la fois pour modéliser des systèmes cyber-physiques, mais aussi pour prouver qu'ils respectent les propriétés voulues. Ainsi, à chaque étape de la conception, on s'assure que les propriétés sont conservées, si possible à l'aide d'un ordinateur. Le système résultant est *correct par construction*.

Plus précisément, nous avons défini formellement une notion de composant dans la logique dynamique différentielle. Un *composant* est constituée d'une partie discrète, par exemple le contrôleur du niveau d'eau dans une cuve, et d'une partie continue –sous la forme d'une équation différentielle–, par exemple l'évolution du niveau d'eau, le tout formant un système cyber-physique. On ne fait pas d'hypothèses de priorité entre chaque partie, ce qui correspond à la réalité de fonctionnement des systèmes cyber-physiques. Le système est réactif, et peut s'exécuter pour une durée de temps arbitrairement longue. Il est possible pour un composant

d'être uniquement discret, par exemple le contrôleur du niveau d'eau, ou uniquement continue, par exemple l'évolution du niveau d'eau. Cette définition est assez générique pour permettre de modéliser un vaste ensemble de systèmes industriels.

Une fois la notion de composant définie, il nous faut un moyen pour les assembler. On a défini un *opérateur de composition parallèle*. Il permet de modéliser deux composants s'exécutant simultanément – en parallèle–, par exemple deux cuves d'eau. Le parallélisme des parties discrètes est obtenue par entrelacement; n'importe quelle ordre d'exécution est possible. Le parallélisme des parties continues est obtenue en considérant le système composée de chaque équation différentielles. Le composant résultant est la réunion des deux parties, encore une fois sans hypothèse de priorité.

Notre opérateur de composition parallèle est syntaxique et peut donc être implémentée par un ordinateur. Ainsi, un ingénieur aurait à définir chaque composant formellement, puis le système résultant par composition serait générée par l'ordinateur, réduisant les risques d'erreurs d'inattention.

Notre opérateur possède deux importantes propriétés algébriques : la *commutativité* et *l'associativité*. La commutativité veut dire que l'ordre de composition n'est pas pertinent. Composer une cuve d'eau `A` en parallèle avec une autre cuve d'eau `B` en parallèle revient exactement à la même chose que de composer la cuve `B` en parallèle avec la cuve d'eau `A`. L'associativité veut dire que l'on peut construire notre système pas-à-pas. On n'est pas obligé de composer tout les composants d'un coup. Ainsi la composition d'une cuve `A` avec une cuve `B`, puis le tout composé avec une troisième cuve `C` est équivalent à la composition des cuves `B` et `C` d'abord, puis composé ensuite avec `A`.

L'associativité est une propriété clé pour obtenir une approche vraiment modulaire. Des travaux précédents avaient déjà proposé une approche compositionnelle dans la logique dynamique différentielle, mais leur opérateur de composition parallèle n'était pas associatif.

On a donc une méthode permettant de modéliser un système cyber-physique de manière modulaire grâce à une approche par composant. Mais ce n'est que la moitié du résultat désiré. On veut aussi pouvoir faire les preuves de correction des systèmes cyber-physique de manière modulaire.

A chacun de nos composants, on associe ce que l'on appelle un *contrat*. Un contrat est une structure formelle permettant de représenter les hypothèses sous lesquelles un composant peut s'exécuter ainsi que les garanties apportées par le composant. On peut prouver ensuite en utilisant le système de preuves de la logique dynamique différentielle qu'un composant satisfait son contrat associé, *i.e.* que sous les hypothèses énoncées, les garanties seront toujours vérifiées pour n'importe quelle exécution du composant.

Nous avons démontré un théorème qui nous assure que sous l'hypothèse que chaque composant satisfait son contrat, la composition parallèle de ces composants satisfait la conjonction des contrats. La conjonction de deux contrats est le contrat où les hypothèses sont la conjonction des hypothèses respectives et les garanties sont la conjonction des garanties respectives. Ainsi, on conserve la satisfaction des contrats à travers la composition permettant d'obtenir la preuve de bon fonctionnement du système par construction. Cependant, plusieurs conditions sont requises pour que ce théorème soit applicable.

La première condition est que les composants ne partagent pas de sorties communes. Cela veut dire que deux composants ne doivent pas agir sur les mêmes parties du système. Par exemple, deux contrôleurs pour des cuves différentes ne doivent pas contrôler la même vanne. Il s'agit d'une bonne pratique de conception, deux composants agissant sur une même sortie amène des conflits et peut produire des comportements inattendus.

La second condition est que les garanties d'un composant ne doivent pas faire référence aux sorties d'un autre composant. Ainsi, les garanties d'un contrôleur d'une cuve ne doivent pas référer aux sorties d'un contrôleur d'une autre cuve. Il s'agit là aussi d'une bonne pratique de conception. Les garanties d'un composant doivent faire référence à ses propres sorties, *i.e.* celles sur lesquelles il a une capacité d'action.

La dernière condition est que les garanties d'un composant ne doivent pas infirmer les hypothèses d'un autre composant. En effet, cela veut dire qu'un composant ne se comporte pas de la manière attendue par le second composant. Il y a donc un problème de conception.

Ces trois conditions sont énoncées de manière syntaxique et peuvent donc être vérifiées par un ordinateur. Ainsi, il est possible d'implémenter complètement cette méthode pour conserver les contrats lors de la composition.

Pour résumer, nous avons proposé une approche permettant de modéliser et vérifier un système de manière modulaire. Nos définitions sont syntaxiques et l'approche peut donc être implémenté afin d'automatiser notre approche.

Nous avons implémenté un prototype dans le prouveur interactif de théorèmes KeYmaera X. Cela permet de montrer la faisabilité d'une telle implémentation. Nous avons étudié deux systèmes industriels afin de valider notre approche. Le premier est un régulateur de vitesse, le second une usine de traitements des eaux composée de plusieurs cuves d'eau en parallèle. L'étude de ces deux exemples a permis de montrer la viabilité de notre approche, mais a aussi mis en relief les améliorations possibles.

Un point particulièrement important concerne la classe des systèmes contrôlés par ordinateur. Il s'agit des systèmes où une grandeur physique, par exemple l'évolution du niveau d'eau d'une cuve, est régulé par un programme, par exemple le contrôleur du niveau d'eau. Le programme surveille l'évolution continue à l'aide d'un capteur qui mesure la valeur périodiquement. Le programme doit ensuite agir de manière à ce que les propriétés désirées soit toujours respectées. Ainsi, pour la cuve d'eau, on veut que le niveau d'eau ne déborde jamais. La caractéristique clé de ces systèmes est la relation périodique au temps. Le contrôleur doit avoir une information suffisamment souvent pour pouvoir réguler le système.

On a montré comment on pouvait améliorer notre approche précédente pour prendre en compte ces aspects temporels supplémentaires. Lors de la définition d'un contrôleur, l'ingénieur doit préciser sa période d'exécution, *i.e.* l'intervalle maximum entre deux exécutions du contrôleur. Lors de la définition de la partie dynamique, il doit fournir sa contrôlabilité. Il s'agit de la durée maximum pendant laquelle le système dynamique peut évoluer sans intervention du contrôleur et encore satisfaire les propriétés voulues. On a modifié notre opérateur de composition parallèle et démontré que l'on conserve la satisfaction des contrats lors de la composition sous les mêmes conditions. On requiert en plus que la période d'exécution du contrôleur soit inférieure à la contrôlabilité du système dynamique. Il s'agit encore une fois d'une bonne pratique de conception, un système ne satisfaisant pas cette condition aurait peu de chance de fonctionner.

On a ensuite généralisé cette adaptation. On associe une période d'exécution lors de la définition d'un composant discret et une contrôlabilité lors de la définition d'un composant continu. On modifie aussi légèrement notre opérateur de composition parallèle pour prendre en compte ces deux caractéristiques temporelles. Lors de la composition parallèle de deux systèmes discrets, la période d'exécution résultante est la somme des périodes d'exécution. En effet, on suppose que l'exécution se fait sur une seule unité de calcul et le parallélisme est obtenue par entrelacement. La contrôlabilité d'un composant continu résultant de la composition parallèle de deux composants continus est le minimum des contrôlabilités respectives.

Malgré ces ajouts, notre opérateur de composition parallèle reste commutatif et associatif.

De la même manière que l'on a généralisé la modélisation par composants à des systèmes avec des caractéristiques temporelles, on a généralisé notre théorème permettant de conserver la satisfaction des contrats lors de la composition. Étant donné la preuve de satisfaction des contrats respectifs, on peut exhiber la preuve de satisfaction de la conjonction des contrats. Les mêmes conditions que précédemment sont nécessaires ainsi que la période d'exécution résultante doit toujours être inférieure à la contrôlabilité résultante.

Finalement, on a adapté notre approche pour modéliser deux autres cas que l'on rencontre fréquemment dans les systèmes industriels : les modes de fonctionnement et la causalité temporelle. L'exemple récurrent des modes de fonctionnement est la présence d'un mode nominal, où le système fonctionne normalement, et d'un mode dégradé, où le système doit garantir uniquement des propriétés vitales. Par exemple, dans le cas de nos cuves d'eau interconnectés, le premier mode correspond au fonctionnement normal où les niveaux d'eau ne doivent pas déborder, mais aussi être suffisamment élevés pour garantir un débit minimal à la sortie de la cuve. Le mode dégradé correspond à une urgence, par exemple un bouton poussoir d'urgence enclenché, et toutes les vannes sont fermées. On veut juste alors garantir que les cuves ne débordent pas.

On modélise chaque mode comme étant un composant et on obtient le système comportant les deux modes de fonctionnement avec notre opérateur de composition parallèle. Mais on ne peut pas conserver les contrats lors de la composition. En effet, nos deux composants représentant chacun un mode ont des sorties communes, et les conditions de notre théorème de composition ne sont pas donc pas remplies. Mais ces deux modes ne s'exécutent pas vraiment en parallèle, ils sont même exclusifs; il n'est pas possible qu'ils se déroulent simultanément. On caractérise chaque mode avec une formule, et sous condition que les formules sont contradictoires, on peut conserver les contrats lors de la composition.

Nous avons aussi défini un opérateur de composition causale. Il est souhaitable parfois d'ordonnancer deux composants plutôt que de les exécuter en parallèle. Par exemple, un capteur doit toujours s'exécuter avant le programme utilisant les données du capteur pour réguler le système dynamique. Dans le cas contraire, le programme raisonnerait avec des données périmées et ne se comporterait pas comme attendu. Ce nouvel opérateur de composition reste compatible avec l'opérateur de composition parallèle et est toujours associatif, nécessaire pour la modularité de notre approche. Comme précédemment, nous avons démontré un théorème permettant de conserver les contrats lors de la composition.

En conclusion, nous avons développé une méthodologie modulaire fondée sur la composition pour modéliser formellement et vérifier des systèmes cyber-physiques. Nous avons illustré qu'elle peut être implémenté et modifié pour s'adapter à de nouveaux défis. Elle sert de base théorique pour développer un outil sur la modélisation formelle et la preuve de systèmes cyber-physiques.

Introduction

Programs are used to govern numerous parts of our society: in the monitoring of railway networks or money transfers for example. It is mandatory that they behave correctly. When we order online goods, we want to be sure that no one can access our personal information. Power grids, water-plant and planes are all supervised by complex programs; a bug may lead to disastrous and potentially deadly consequences. Such systems are called *safety-critical*.

To avoid bugs, software engineers have developed numerous testing procedures, but they are fundamentally limited. "The test of programs may be a very efficient way to show the presence of bugs, but is desperately inadequate to prove their absence" (Djikstra, 1972). A test can not ensure that every behavior of a program is correct since there is an infinity of behaviors, it only assesses that it works for the finite set of tested behaviors. Another approach is to use the so-called *formal methods*. Instead of testing if a program works correctly for some behaviors, it advocates the use of mathematics to prove that the program behaves correctly for every possible execution. They are costly, being thus applied for critical programs only.

Numerous approaches and tools have been developed these past decades, and great successes have been achieved. The CompCert compiler is a compiler for a consequent subset of the C language formally verified using the theorem prover Coq. The Communication-Based Train Control (CBTC) program running the lane 14 of Parisian subway has been entirely developed using the B method, an other formal verification technique. But they have been obtained at the cost of great effort and intensive research, and their systematic replication in the industry is still not feasible.

Most approaches to the problem of verification of program assumed it to be already written and try to prove that it behaves correctly. The conception of the program is usually achieved by an expert of the domain and the verification process by a proof engineer who is not familiar with the domain. The former conducts the development without thinking about the formal verification. The latter has to deconstruct the software in order to conduct the proof. It results in a huge amount of wasted time and resources, and possibly misunderstandings. The *correct-by-construction* approach advocates the idea that the design and the verification should not be separated, but rather conducted simultaneously. The verification provides significant insights for the development of the software which is in return produced with its verification as a goal. The B method follows this paradigm, which is believed to be the key point of its success.

In the case of the CBTC verification, only the program has been verified; the physical quantities involved (*e.g.* the speed of the train) are not considered or idealized. Programs as the CBTC do not simply perform a computation, they exhibit frequent interactions with the physical environment. The resulting behavior can not be fully understood without taking physical evolution into consideration. Since such programs are often critical, *e.g.* auto-pilot of planes or monitors of water-plant treatment, it is highly desirable to develop *specific* formal methods.

Systems where physical behavior, *e.g.* the speed of a train, is mixed with discrete behaviors, *e.g.* the CBTC, are called *cyber-physical systems* (CPS) or *hybrid systems*. The first behavior assumes a continuous model of time when the second possesses a discrete model of time. The difference of nature between theses two models render the formal design of hybrid systems very complex. Plus, many systems exhibit intangible interactions between discrete and continuous behavior. Reasoning about such systems demands tools capable to accommodate the different abstractions level such as the system level, processor level, logic level, etc.

In 2007, the Differential Dynamic Logic is defined by André Platzer to model and verify

such systems. It provides a precise mathematical semantics and considers physical evolution on the same level as programming constructs. It features a proof system to ensure the correctness of behaviors which has been used to prove numerous use-cases, some of them unverified previously. The proof system has been implemented in the interactive theorem prover KeYmaera X. This theorem prover is designed toward automation which is a key functionality to make tractable proofs.

Despite this, most of the problems are still intractable in practice and a methodology to tackle large problems is thus required. One of the most ancient method to address a difficult problem is to divide it into smaller parts, as exposed by Descartes in his book *Discours de la méthode*. Each part is easier to understand for a human, but also easier for a computer to treat. Seeking methods to efficiently break down a system is thus highly desirable. Numerous works have been produced for programs, but few have been done for cyber-physical systems and in Differential Dynamic Logic.

My main contribution is the definition of a modular component-based framework in Differential Dynamic Logic to model and prove correctness of Cyber-Physical Systems. It provides theoretical basis to represent parallel composition of Cyber-Physical systems and how to implement a correct-by-design approach.

Contents

1 Adressed problem **15**

1.1 Verification of Cyber-Physical Systems 16

1.1.1 Scalable verification of hybrid systems 16

1.1.2 Interest of hybrid systems 16

1.1.3 Our approach: correct-by-design 17

1.2 State of the art 17

1.2.1 Formalisms for discrete systems 17

1.2.2 Addition of continuous features 18

1.2.3 Other approaches to verification of Cyber-Physical Systems 21

1.2.4 Existing tools 23

1.3 Contributions 25

2 Introduction to Differential Dynamic Logic **27**

2.1 Modeling of Cyber-Physical Systems 28

2.1.1 Discrete behaviors or programs 28

2.1.2 Ordinary Differential Equations 31

2.1.3 Blending discrete and continuous aspects 33

2.2 Expressing Properties in $d\mathcal{L}$ 33

2.2.1 First-order Real Arithmetic 34

2.2.2 A modal logic 35

2.3 Proving properties in $d\mathcal{L}$ 36

2.3.1 Generalities on sequent calculus 36

2.3.2 First Order Real Arithmetic 37

2.3.3 Structural rules 41

2.3.4 Hybrid program rules 42

2.4 Theoretical results 51

3 A modular component-based approach in Differential Dynamic Logic **53**

3.1 Definition of a component 54

3.1.1 What is a component 54

3.1.2 Definition in $d\mathcal{L}$ 58

3.2 Parallel composition operator 61

3.2.1 Parallel composition of components 61

3.2.2 Definition in $d\mathcal{L}$ 63

3.2.3 Comparison with the parallel composition of hybrid action systems . . 67

3.2.4 Relation to the meta-theory of Benveniste 69

3.3 Modular proof . . . 70
3.3.1 Necessary conditions . . . 71
3.3.2 Technical result . . . 73
3.3.3 For two continuous components . . . 74
3.3.4 For two discrete components . . . 82
3.3.5 For a discrete component and a continuous component . . . 85
3.3.6 For two general components . . . 89
3.4 A prototype in KeYmaera X . . . 92
3.4.1 Presentation of Bellerophon . . . 92
3.4.2 Implementation in KeYmaera X . . . 93
3.5 Study of a water-plant example . . . 100
3.5.1 Environment and initial conditions . . . 100
3.5.2 Water-level . . . 102
3.5.3 Controller . . . 103
3.5.4 Water-tank . . . 104
3.5.5 Water-Plant . . . 107
3.5.6 Discussion . . . 108

4 Extensions of our framework **110**
4.1 Computer-Controlled Systems . . . 111
4.1.1 Modeling Computer-Controlled Systems . . . 111
4.1.2 Coverage of the standard encoding . . . 114
4.1.3 Modular proof of a Computer-Controlled System . . . 116
4.2 Parallel composition in a timed framework . . . 118
4.2.1 Definition of a timed component . . . 119
4.2.2 Timed parallel composition for discrete components . . . 121
4.2.3 Timed parallel composition of continuous components . . . 124
4.2.4 Timed parallel composition of a discrete and a continuous component 127
4.2.5 Timed parallel composition of two general components . . . 129
4.3 Handling of modes . . . 133
4.3.1 Modular modeling . . . 133
4.3.2 Modular proof . . . 135
4.4 A Causal Composition operator . . . 139
4.4.1 Modular Modelling . . . 139
4.4.2 Algebraic properties . . . 141
4.4.3 Modular proof . . . 143

5 Future works **146**
5.1 Continuing from the results on parallel composition . . . 147
5.1.1 Extending the parallel continuous composition with complementary domains . . . 147
5.1.2 Relaxing conditions of composition theorem . . . 149
5.1.3 Addition of communication channels . . . 150
5.1.4 Implementation . . . 153
5.2 Improving extensions of the parallel composition . . . 154
5.2.1 Timed parallel composition with several CPUs . . . 154
5.2.2 Modes for continuous systems . . . 154

5.2.3 Causal composition for continuous components 155
5.3 Toward integration of refinement into component-based approach 155
5.3.1 Refinement in $d\mathcal{L}$ 155
5.3.2 Refinement and parallel composition 156

Chapter 1

Adressed problem

Cyber-Physical systems (or hybrid systems) are pervasive in our society. Autonomous vehicles, water-plant or train control systems are examples of such systems. It is important to have methods to faithfully model such systems and soundly reason on it. Furthermore, these methods should be scalable, *i.e.* applicable to industrial systems as a water-plant factory.

We present briefly the problem of the scalable verification of hybrid systems in Section 1.1. Verification of hybrid systems is an important subject and numerous solutions have been proposed. We present in Section 1.2 such solutions. Lastly, we detail our contribution in Section 1.3; it consists of a modular component-based framework, to address the problem of modeling and proof of hybrid systems.

1.1 Verification of Cyber-Physical Systems

In Subsection 1.1.1, we give a detailed presentation of hybrid systems and of the verification problem. We identify the challenges specific to this problem. We justify in Subsection 1.1.2 the interest of a formal treatment of hybrid systems and detail the need for proved systems. In Subsection 1.1.3, we detail briefly our approach.

1.1.1 Scalable verification of hybrid systems

Cyber-Physical Systems (CPS) (or hybrid systems) mix continuous behaviors with discrete behaviors. Continuous behaviors can be the speed of a vehicle or the water-level in a tank. Discrete behaviors can be computer programs as a tachymeter or a water-level controller. It can also be discontinuities between two continuous behaviors. In the bouncing ball example [105], the fall and the bounce of the ball are two different continuous evolution and the transition is modeled by a discrete behavior.

The problem of verification of hybrid systems consists in the development of mathematical methods to ensure the correct behavior of such systems. Numerous methods have been proposed over the years for cyber systems, but much less for CPS. The reason is the complicated interaction between the continuous behaviors and discrete behaviors. They have a distinct model of time. Computers are assumed to function on a discrete basis. Between two operations, there is a gap. But a plant evolves continuously; the speed of a car does not jolt.

Numerous solutions have been proposed to answer the challenge of modeling and verification of CPS. But they are still not applicable to industrial systems as a water-plant factory due to an increasing complexity in the conception or/and the verification. There is a need for methodologies to scale up.

1.1.2 Interest of hybrid systems

Hybrid systems are ubiquitous and are used daily by everyone without knowing it. They perform numerous tasks, most of them being critical. It is thus mandatory to ensure that they behave correctly, *i.e.* they meet their requirements. The most common methods to verify it is the use of testing methods. But they are fundamentally limited since they can only show the presence of bugs, not their absence. Formal methods ensure such absence and are thus desirable for systems where the failure is not acceptable.

A massive amount of efforts have been put to the development of efficient verification tools and of methodologies for computer systems, but few for cyber-physical systems due to the complicated relation between continuous and discrete behaviors. Efficient verification tools

reduce the proof effort to ensure that desired safety properties are satisfied. Methodologies provide designers with guidance to efficiently model and prove systems. There is a crucial need for efficient tools and methodologies to scale formal verification for hybrid systems.

1.1.3 Our approach: correct-by-design

The current process to obtain reliable cyber-physical systems is to first design the system, then to verify if meets its specification. But, once a system is designed, it is very difficult to prove its correctness with respect to the specification. The processing of a proof is very different from the design process and is often realized by a different person or team. Communication issues add up in the case of different teams restraining more an efficient processing of the proof. It is thus highly desirable to have methods to carry the design phase simultaneously with the verification phase. It is the correct-by-design spirit.

Numerous formal methods have been developed to ensure correctness of a system, but a few of them are scalable, *i.e.* that can be applied to realistic systems used in the industry and not just on toy examples. The correct-by-design approach has been used for most of the realistic systems verified by formal methods, *e.g.* the lane 14 of the Parisian subway. But it is not enough to ensure scalability and the other proven approach is the component-based approach.

1.2 State of the art

The most popular approach to tackle the issue of modeling and verification of hybrid systems has been the extension of existing formalisms with hybrid features, *i.e.* with differential equations. We present such formalisms and their extensions in the two Subsections 1.2.1 and 1.2.2. Yet, several formalisms are not adaptation of previous work. We present them in Subsection 1.2.3 s.

1.2.1 Formalisms for discrete systems

We present briefly formalisms primarily introduced to model and verify computer programs or discrete systems. They have been extended to hybrid systems.

Automata theory *Automata theory* denotes the approach where systems are modeled by automata and the verification is performed by model-checking [68]. It is widely used to model formal languages and plays a major role in the field of formal verification.

B/Event-B The *B-method* is a refinement-based approach to develop programs from an abstract specification [3]. It has been successfully used to develop the lane 14 of the Parisian subway.

Event-B is an extension with a more flexible approach to refinement [70]. It is for example possible to introduce events during a refinement step.

Dynamic Logic *Dynamic Logic* is a deductive approach to the verification of infinite-state discrete systems [108]. Its characteristics are to internalize the operational model of the system within logical formulas to the same level as the properties to be verified. A single

formula thus represents a system with its environment, assumptions, model and guarantees of good behavior. It has been able to verify consequent Java programs [91].

Action Systems *Action Systems* are a formalism describing general reactive systems in terms of atomic actions occurring during the execution of a system. The approach has been applied to parallel and distributed systems [13].

It allows to model terminating or infinitely repeating systems, *e.g.* embedded systems. Since most of hybrid systems are reactive systems, several extensions to continuous behaviors have been proposed.

Communicating Sequential Processes *Communicating Sequential Processes* (CSP) is a formalism proposed by Hoare in 1978 [66] to describe concurrent aspects of programs. CSP allows to describe systems as independent component processes. It uses message-passing communication to model the interaction between processes. It is mainly used to model safety-critical systems.

Synchronous languages *Synchronous languages* have been developed to ensure the correctness of safety-critical embedded systems [21,53,58]. They support functional concurrency and synchronicity. They aim to remain simple in order to facilitate the adoption by software engineers. They are founded on a solid mathematical basis which makes it amenable for proofs of correctness and certification purposes. Several successes in the industry have been obtained [19].

1.2.2 Addition of continuous features

This section presents formalisms obtained by the addition of continuous features. The original formalisms are presented in Subsection 1.2.1. The approach consists of the adjunction of differential equations as a new language construct.

Hybrid automata *Hybrid automata* is a formalism introduced in 1993 to model hybrid systems. It extends the notion of automata with differential equations [8,63]. It has been one of the first formalisms for modeling systems with mixed discrete-continuous behavior and has attracted a lot of attention during the last decades. Hybrid automata have allowed the study of numerous sub-classes of hybrid systems and provided important theoretical results.

Once a system is modeled as a hybrid automaton, we verify properties by model-checking. The properties are usually expressed in the Real Arithmetic theory. For example, the model-checker Hytech can verify properties on linear hybrid automaton [65].

Given two hybrid automata, it is possible to compose them in parallel to obtain a new system. The resulting hybrid automaton is exponential in size, and thus intractable; it is the *state-space explosion problem*.

I/O hybrid automata *I/O hybrid automata* is an extension of hybrid automata where the inputs and outputs are explicit [83]. It has been developed to tackle the composability issue and it allows to use the notions from contract theory. The verification of systems is also performed by model-checking of desired properties.

Simulation-based verification Verification by simulation [40, 41, 54, 60, 90] considers full set of time-bounded trajectories of a hybrid system evolving from an initial state. The verification is carried on by means of finite sample of initial states and a sensitivity argument. It starts with a sufficiently dense sample of initial states on which numerical simulation is applied to obtain the corresponding trajectories. The sensitivity argument is then used to over-approximate the "tube" of trajectories.

Verification of stochastic hybrid systems Stochastic behaviors are omnipresent in hybrid systems due to the inherent uncertainty of environment or simplifications tackle the complexity of modeling and verification. Stochastic hybrid systems exhibit a tight interaction of discrete, continuous and stochastic behaviors.

Hybrid automata have been extended to model such systems. The verification is then done through reachability analysis [2, 7, 29, 44, 121, 129] or simulation [84, 133]. Theses works have introduced different notion of hybrid automata which differ on the randomness is introduced.

Another possibility is to introduce stochastic differential equations which generalize differential equations [2, 30, 69]. A compositional modeling framework have been proposed to handle the modeling of complex stochastic hybrid systems by André Platzer in $d\mathcal{L}$ [100] and the formalism HMODEST [57]. This last works extends the MODEST modeling language with differential equations. MODEST is high-level language inspired by process algebra and features compositional modeling as a native feature.

Hybrid Event-B *Hybrid Event-B* intends to apply the approach by refinement, characteristic of the B method, to hybrid systems [4, 16]. The development is carried as with Event-B, except that continuous behaviors can be specified.

Differential Dynamic Logic ($d\mathcal{L}$) *Differential Dynamic Logic* ($d\mathcal{L}$) is a hybrid extension of the Dynamic Logic. It is a logic-based approach to the modeling and verification of hybrid systems [95]. The theory is implemented in the theorem prover KeYmaera X [85] whose core has been formally verified in Coq and Isabelle [25].

Differential Dynamic Logic provides important theoretical results. It has proved that discrete systems, continuous systems and hybrid systems are equivalent in a proof-theoretic viewpoint [101]. More precisely, there is a sound and complete axiomatization of hybrid systems relative to continuous dynamical systems, and conversely, there is a sound and complete axiomatization of hybrid systems relative to discrete dynamical systems. These results increase the confidence in the ability of computers, intrinsically discrete, to handle the verification of hybrid systems.

Hybrid systems are modeled by the so-called *hybrid programs*. The discrete fragment is a small and Turing-complete programming language. It corresponds to the modeling part of Dynamic Logic, which has been successfully used to verify several Java programs in the theorem prover KeY [6].

The continuous fragment is comprised of *Ordinary Differentials Equations* (ODEs). The instructions of the programming language are at the same level as discrete instructions. They can thus be combined freely with them. The differential equations are required to have a solution, but it is not mandatory to exhibit it to verify properties.

Properties are expressed in First-Order logic of Real Arithmetic augmented with a modality ($[.]$). Real Arithmetic allows to represent polynomials, but not trigonometric or exponen-

tial functions. First-Order Logic connects real arithmetic terms. The modality articulates the behavior of hybrid programs with the properties.

A proof system under the form of a *sequent calculus* has been defined. Key aspects of sequent calculus are a syntactic deductive approach amenable to automation and an adaptability to extensions of logic. It features rules to handle First-Order logical connectives and structural properties of proofs and rules to reason on hybrid programs. The former are common to many other sequent calculus. The latter can be subdivided in two. One part is devoted to discrete instructions, and corresponds to rules in Dynamic Logic. The second part regroups rules to reason on ODEs. Among these, the rule of Differential Invariant stands out by its importance. It allows to prove that an ODE satisfies a property without having to provide the solution. It makes a parallel between the discrete iteration and an ODE. The sequent calculus has been proved *sound*, *i.e.* that every derived formula is semantically true.

Several extensions of $d\mathcal{L}$ have been proposed. In [96] and [71], the authors proposed an extension of $d\mathcal{L}$ with the diamond – $\Diamond$ – and the box – $\Box$ – constructs of standard temporal logic [107]. This conservative extension allows to express that a property φ is true along all states of every trace of a hybrid program α – $[\alpha]\Box\varphi$. Other extensions are Quantified Differential Dynamic Logic used to model distributed hybrid systems [99], or an extension for hybrid games [109]. In [118], the authors introduce architectural abstractions for hybrid programs. They address the issues caused by the monolithic approach of $d\mathcal{L}$, difficulty of understanding and change, bu using component-based engineering.

The proof system has been first implemented in KeYmaera [105], which is an extension of the KeY theorem prover [17]. The theorem prover has been completely rewritten to become KeYmaera X [47]. The kernel asserting and checking the correctness of the proof has been reduced to several thousands of lines of code and has been verified in Coq and Isabelle [25]. KeYmaera X is aimed toward automation [85]. It uses back-end tools, Z3 [39] and Mathematica [126], to respectively discard real arithmetic formula and reasoning on differential equations.

In [88], the authors introduced a commutative (non-associative) composition operator in $d\mathcal{L}$. It allows to break down a system model into independent functional parts or components. Under a proof of the properties associated to these components as a contract, a theorem allows to transfer properties to the global system. A more recent work of A. Muller and A. Platzer [89] is more closely related to our contribution. It extends previous work on component-based design in $d\mathcal{L}$ by presenting a way to handle retro-action along with a methodology to efficiently use it. It adds an important feature to earlier work [88] allowing to model a wider class of systems, but it still lacks modularity for design and proof automation capabilities as the composition is not associative.

Hybrid Action Systems Two different extensions of Action Systems have been introduced: Differential Action Systems and Continuous Action Systems.

Differential Action Systems are an extension of Action Systems with a new action: the differential action [114, 116]. It represents differential equation. The weakest precondition reasoning associated for differential action is developed with it.

The approach taken by *Continuous Action Systems* [12] is to enhance the variable with time-dependent attributes. They are seen as functions from $\mathbb{R}_+$, the time-domain, to continuous or discrete value domains. It allows to model real-time systems without adding new actions and thus not having to redesign a proof theory behind it.

Hybrid Communicating Sequential Processes *Hybrid Communicating Sequential Processes* is an extension of the process algebra CSP proposed in 1994 by Jifeng He [32, 73]. It integrates real-time and continuous constructs such differential equations to model continuous evolution. As a process algebra, HCSP features to construct complex systems out of simpler ones, notably using communication channels and a parallel composition operator which are native. It has been used to model several industrial systems such as the Chinese Train Control System [80] and a running aircraft [124].

In 2010, a Hoare-style calculus is introduced to reason about HCSP [56,80,123]. It makes an extensive use of the Duration Calculus [31] to record the execution history of HCSP process. The reasoning is a standard pre and post-conditions calculus. To reason about continuous evolution, the authors introduce differential invariant [104, 120]. They have extended their approach to take into account communication failures, or probability and stochastic behaviors [93, 124]. The calculus have been implemented in the theorem prover Isabelle [92] using both a shallow and deep embedding [125]. They make use of the SledgeHammer tool to automate consequent parts of the proof. They have used it to verify several industrial systems such as the Chinese Train Control System [80, 123] or the descent guidance control program of a lunar lander [130].

A framework to link HCSP models to Simulink diagrams have been proposed Zou *et al.* [132]. Simulink diagrams are prominent in the industry, and brings many benefits to validate systems by simulation, but they lack formal guarantees. Translating a Simulink model into an HCSP model allows to obtain them. They have extended it to take into account Stateflow diagrams [131].

Yan, Zhan *et al.* show how to discretize continuous HCSP to discrete HCSP. From the discrete model, they generate executable code and prove that the generation does not alter the meaning of the program [127, 131]. It leads to a toolcgain, MARS [33], within which it is possible to model a hybrid system with Simulink, translate it into an HCSP process, generates invariant and verify them using the implementation into Isabelle, and finally obtain executable SystemC code [128].

Extensions of synchronous languages Zelus is an extension of Lustre with Ordinary Differential Equations [26]. It allows to model both continuous and discrete behaviors and stay compatible with synchronous programs.

1.2.3 Other approaches to verification of Cyber-Physical Systems

Other approaches have been proposed and are not the extension of a formalism with differential equation. They may be a new formalism as the δ-decidability. They may be also formalisms which have not been primarily intended for hybrid systems, but is expressive enough to be applied to, *e.g.* TLA or Coq.

δ-decidability *δ-decidability* has been introduced by Sicun Gao, Jeremy Avigad and Edmond Clark in 2012 [49, 50]. It is a new decision procedure to decide the satisfiability of formulas of extensions of Real Arithmetic. Real Arithmetic is decidable, but extensions with trigonometric or exponential functions are undecidable. This undecidability holds for a precise and symbolic decision procedure. Most of the algorithms used for numerical computation are based on approximations and are precise only up to a certain error. The authors follow

this idea to define a *relaxed* notion of correctness up to some error δ, hence the name of δ-decidability.

They define a procedure called the *δ-decision problem.* It either proves that the system is safe or states that it is unsafe under some perturbation which can be made arbitrarily small. It is implemented in the SMT solver dReal [51]. The procedure is fully automatic. It has been adapted for several extensions of the theory and provides a way to encode ODEs [52]. A system has to be modeled as one formula in the SMT solver dReal, and it is thus not compositional.

Linear Temporal Logic (LTL) The *Linear Temporal Logic* (LTL) [107] is a modal logic devised to reason about temporal properties of a program. It allows to state properties such as "For every execution of the program, the property φ will hold" or "There is an execution of the program for which the property φ holds". A generic use is to design a system under the form of an automaton, and then to model-check a property expressed in LTL [34, 35].

It is widely used to express properties of discrete fragment of hybrid systems. But it is not adapted to reason about complete hybrid systems which exhibit continuous behaviors since the properties of LTL are related to discrete executions of a program.

Temporal Logic of Actions (TLA and TLA+) *Temporal Logic of Actions* (TLA) is a formal specification language inspired by the Temporal Logic of A. Pnueli [107], and devised by L. Lamport in 1990 [76]. The author intend to provide a mathematical setting to specify concurrent programs.

The extension *TLA+* is designed to be more suitable for engineers by providing means to represent large formulas and systems [77]. Both formalisms feature a proof system to perform proofs. It has been used to specify hybrid systems in [110], and a general approach to the verification of hybrid systems is proposed in [76].

Modular proofs of hybrid systems in Coq In [111, 112], the authors follow the idea of modular proof of hybrid systems in the foundational proof assistant Coq. They have been able to verify the behavior of a drone.

Hybrid Algebras for hybrid systems Several attempts have been made toward the definition of a process algebra for hybrid systems. Brinksma and Krilavicius propose an extension of the classical processes algebra to hybrid processes [27, 28], but their approach stays at the design level and does not provide any tool to prove properties. Peter Höfner presents algebraic approaches for several formalisms such as hybrid automata, CTL or Neighborhood Logic [67]. R. Alur *et al.* have presented a theory of modular design and refinement from hierarchical state machines implemented in Charon [9], where verification of large systems would amount to non-modularly explore the state-space of its composed elements.

Abstract interpretation *Abstract interpretation* tackles the problem of verification of properties on a complex system by abstracting it [37]. It aims to formalize the notion of approximation. The system is transferred into an abstract domain where it is easier to verify properties. The proof on the abstract domain transfers to the concrete domain. This approach has obtained some great successes [38]. Several specific abstract domains have been proposed for hybrid systems [10, 59].

Contract-based design *Contract-based design* advocates the use of *contracts* to precisely define the inputs and outputs of a system. In the legal system, a contract formalizes the commitment of a supplier, an employee or a product, for example the supply of ten tons of steel per day to a car company by a smelting plant, the selling of five cars per day or the guarantee that the product will work perfectly fine for three years. It formalizes also the assumptions under which the output is guaranteed, for example, there is enough iron ore delivered to the smelting plant, that the employee is provided with a computer and a salary, or again that the product is used properly.

Contracts in systems are pairs of formulas (A, G) which state the *assumptions* A under which the system operates, and its outputs satisfy the *guarantees* G. A and G are formulas of a logic defined according to the type of system under consideration. It is known also as *pre* and *post-conditions* and is used in several proof tools as [43].

The meaning of a contract is intuitive to grasp and matches the current work-flow of engineers. It is natural to use it with a component-based approach. We associate to every component a contract representing the assumptions on its inputs and the guarantees on the outputs.

It has attracted a lot of attention in the field of embedded systems and hybrid systems because it is easily understandable by engineers and scalable. But a drawback is that it considers component as a black box, and just specifies the behaviors on inputs and outputs. To develop a system, we need to couple it with a modeling language to specify components, *e.g.* a Java program. To ensure that the properties defined in the contract are coherent and that the component satisfies them, we need a proof system. It can be a theorem prover as Coq [22] or [92] or an SMT solver as [39] or [51].

Benveniste *et al.* defined in 2012 a meta-theory of contracts to provide a unified framework for contract-based design [18]. It can be instantiated by different contract theories, *e.g.* *A*-*G* contracts or *I*/*O* contracts, and aims to ease the communication between separate teams of engineers.

1.2.4 Existing tools

This section is devoted to tools that have been developed to model and verify hybrid systems. There are general interactive theorem provers (Coq, Isabelle) or specific (KeYmaera, KeYmaera X). We present also model-checkers like HyTech, PHAVer or UPPAAL, and SMT-solvers like Z3 or dReal.

General theorem provers: Coq and Isabelle Coq and Isabelle [22, 36, 92] are generic interactive proof assistants first released respectively in 1986 and 1989. Coq implements the Calculus of Inductive Functions. It is used for the formalization of mathematics and proof of mathematical theorems, but also to model and verify software. It provides interactive proof methods along with a tactic language. It has notably been used for the formalization of the proof of the four-color theorem [55] and the verification of a C compiler [79].

Isabelle is also an interactive theorem prover providing a meta-logic used to express mathematical formulas in a formal language. It features several proof tools and has been used for the proof of correctness of the seL4 microkernel [74].

KeYmaera KeYmaera is the first theorem prover to implement $d\mathcal{L}$ [105]. As the Differential Dynamic Logic is an extension of Dynamic Logic, KeYmaera is an extension of the theorem

prover KeY which implements Dynamic Logic. It is coupled with Mathematica [126] and Z3 [39] to treat real arithmetic formulas.

It provides some automation, but is not designed toward this goal; most of the proofs require a manual effort. The kernel amounts to more than one hundred of thousand lines of codes and has not been verified. The proofs produced are not completely trustworthy.

KeYmaera has been completely redesigned into KeYmaera X.

KeYmaera X KeYmaera X is the successor of KeYmaera [47], but completely rewritten with clean foundations and automation in mind. It still uses Mathematica and Z3 as external solvers to discharge goals. The kernel numbers less than ten thousand lines of codes and has been formally verified in Coq and Isabelle [25]. A tactic mechanism, Bellerophon, has been added, which allows proof programming [46].

dReal dReal is an SMT solver specialized in Real Arithmetic enriched with trigonometric functions or exponential functions [51]. It implements the mechanisms of δ-decidability. It has been used to verify hybrid systems that were outside of the scope of existing tools.

Z3 Z3 is a general SMT-solver developed by Microsoft [39]. It is used as a back-end tool in KeYmaera and KeYmaera X to discharge real arithmetic formulas.

HyTech HyTech is a symbolic model checker for hybrid systems [65]. The systems are represented under the form of hybrid automata and safety or liveness properties are model-checked against it. It has been used to verify multiple case studies such as the generic railroad crossing [62, 64].

PHAVer PHAVer is another model checker for hybrid systems [45]. It uses the same core algorithm as HyTech, but adds several features to treat systems that were outside the capabilities of HyTech. It has been used to verify the navigation benchmark [42] or the bouncing ball system.

UPPAAL UPPAAL is a tool designed for the verification of real-time systems, but also modeling and simulation [78]. The systems are modeled as a network of timed automata. It has been used to verify multiple use-cases such as a collision-avoidance protocol [5, 72].

Conclusion The formalisms in Section 1.2.2 and 1.2.3 have all proposed methods to model hybrid systems and means to verify some properties on them. Several ideas have been implemented and tested on various kinds of problems, allowing to evaluate their power. We have presented different verification and modeling tools in Section 1.2.4.

To our understanding, there are two kinds of complexity that researchers have tried to tackle, *mathematical* and *systemic*. The mathematical complexity is about the kind of systems that are considered. Some formalisms are restricted to the linear differential equations and other handle larger classes of ODEs. The properties may be expressed only in Real Arithmetic, or others use trigonometric or exponential functions. From this point of view, $d\mathcal{L}$ and the δ-decidability approach provide great advances.

The systemic complexity refers to systems that do not possess theoretical mathematical difficulties, but where the size, often through repetition of similar components, is the main

difficulty to handle. This problem is not restricted to hybrid systems and is encountered in diverse domains. For example, a factory is often a system with numerous captors, monitors, and actuators, each one being very simple to understand and handle, but the overall behavior is difficult to apprehend.

Two methods have emerged as an efficient way to tame systemic complexity: refinement and component-based approach. The former consists of starting with a simple system, easy to understand and verify. It is upgraded progressively, *refined*, until obtaining the desired system. Each upgrade is kept simple allowing to handle the growing complexity. This approach obtained successful results and the B method is its most famous representative. The latter tames the complexity by breaking a system into pieces, the so-called *components*, which are simple enough to be handled separately. It is a popular approach in computer science or engineering.

1.3 Contributions

Modular component-based approach in Differential Dynamic Logic We have defined a modular component-based approach to model and prove correctness of cyber-physical systems in Differential Dynamic Logic ($d\mathcal{L}$). The modeling and the proof are carried together following correct-by-design principles.

We have defined a notion of *component* in $d\mathcal{L}$ which is able to model a wide variety of hybrid systems. We have defined a *parallel composition operator* to build a system from its parts and illustrated it with a cruise-controller example.

We have proved that the operator is *commutative*, which means that the order of composition is not important, and *associative*, which means that we can compose in a modular way. It allows to model a system by considering each component separately rather than in a monolithic way. To our knowledge, it is the first syntactic parallel composition operator in $d\mathcal{L}$ which is commutative and associative. Previous attempts do not yield associativity, and therefore modularity.

We have associated to every component a *contract* which specifies its assumptions and guarantees. The user is required to prove that the components satisfies its contract using the sequent calculus of $d\mathcal{L}$.

We have demonstrated a *composition theorem* which ensures that we retain contracts through parallel composition; if two components satisfy their respective contract, then the component resulting from their parallel composition satisfies the conjunction of contracts. It allows to carry the proof of correctness of components during the construction of the system, reducing the proof effort to smaller systems, more likely to be tractable. Instead of proving the correctness of the complete system as in the monolithic approach, we have to prove the correctness of each part.

The proof that we retain contracts from composition is constructive; given the proof of satisfaction of respective contracts, we have showed how to obtain a proof of satisfaction of the conjunction of contracts. We have implemented this process as a prototype in KeYmaera X to show the feasibility of an automation of our process. We have studied a water-plant use-case to identify improvement points, leading us to new developments.

Extension of our modular component-based approach We have adapted our parallel composition operator to modularly model *Computer-Controlled Systems* (CCS). They

are systems where a continuous part, the *plant*, is regulated by a program, the *controller*, and regroups most of industrial systems. A key aspect of such system is the timing aspect; the controller must executes sufficiently often to ensure a correct functioning. But our parallel composition operator does not provide insights on the timing aspects.

We have presented how to systematically take into account timing aspects: the *control period* of the plant and the *execution period* of the controller. We have identified a framework to apply our parallel composition operator and demonstrated that we retain contracts through composition, all illustrated with a water-tank example.

We have extended the timing approach to handle systems where several plant and monitors are running in parallel. We keep track of the control period and execution period through composition to ensure it does not result in flawed systems. We have demonstrated that we still retain contract and illustrated how it applies to the water-plant example.

We have identified a framework to soundly and modularly represent *modes* in a cyber-physical systems via an composition operator adapted from the parallel composition operator. As previously, we can associate contracts and retain them through composition.

We have defined a *causal composition operator* to model the ordering of two components, *e.g.* a sensor before a monitor. It is an adaptation of the parallel composition operator and is thus compatible with it. We have showed that it is associative, hence modular, and that we retain contracts through composition.

Chapter 2

Introduction to Differential Dynamic Logic

Differential Dynamic Logic ($d\mathcal{L}$) is an extension of the Dynamic Logic, DL, of V. Pratt [61, 108] obtained by the addition of Ordinary Differential Equations (ODEs). DL is a modal logic developed to model and verify imperative programs, in particular Java programs [6]. $d\mathcal{L}$ extends it in both aspects; it allows the modeling of ODEs and interactions with regular programs, and it provides verification methods for hybrid systems.

$d\mathcal{L}$ has been proposed to model and verify hybrid systems. The dominant approach in the previous decade was the modeling of a hybrid system as a hybrid automaton and the model-checking of various properties. $d\mathcal{L}$ adopts a deductive approach to the problem of verification. It is designed to be a general framework for hybrid systems.

Hybrid systems are represented by so-called *hybrid programs* which can be discrete or continuous. Every system that can be represented under the form of a hybrid automaton can be represented by a hybrid program [95]. It allows the modeling of a wide variety of use-cases. We can model the speed of a car or its position relatively to a fixed point, essential for autonomous vehicles. We can also model the water-level of a tank in a water-treatment plant along with their control systems. An important aspect is the possibility to represent and reason on real time instead of a discretized version.

The properties are formulas of the First-Order Real Arithmetic augmented with the modality $[.]$. It allows to reason on the executions of a hybrid program. For example, we can express that at every point of time, the speed of a car stays in a safe range.

The verification of properties is achieved by a specific *sequent calculus*. It has been demonstrated sound and is implemented in the theorem prover KeYmaera X [47]. It features specific rules to reason on instructions of the language.

We present in Section 2.1 how to model hybrid systems with the help of *hybrid programs*. We show in Section 2.2 how to associate properties to hybrid systems, notably *safety* and *liveness* properties. The Section 2.3 is devoted to the presentation of the sequent calculus and its rules to prove properties on hybrid programs. We present theoretical results on $d\mathcal{L}$ in Section 2.4.

2.1 Modeling of Cyber-Physical Systems

Hybrid systems exhibit both discrete and continuous behaviors. This duality is expressed in $d\mathcal{L}$ by the modeling of ODEs and programs. We present in Subsection 2.1.1 how we model programs and in Subsection 2.1.2 how we represent differential equations. The Subsection 2.1.3 is devoted to examples of hybrid systems modeled in $d\mathcal{L}$.

2.1.1 Discrete behaviors or programs

In this section, we define the syntax and the semantics of discrete programs. We illustrate the semantics with graphical examples and show how to implement the Fibonacci sequence and a car-controller example.

Programs are usual computer programs. They are defined from a small language which is Turing-complete; every program written in C or Java can be expressed in $d\mathcal{L}$. It is easier to reason on a small language, and the lack of usability can be overcome by definition of macros. This small language is the same as in Dynamic Logic (DL) [108] [61].

Definition 1 (Syntax of discrete programs)**.**

$$\alpha, \beta ::= x := \theta \mid ?\varphi \mid \alpha; \beta \mid \alpha \cup \beta \mid \alpha^*$$

where φ is a formula (Def. 6) and θ is a real arithmetic term (Def. 5).

$x := \theta$ is the assignment of term θ to the variable x. $?\varphi$ tests if the formula φ is true. The operator ; denotes the sequence between two hybrid programs and $\cup$ is the non-deterministic choice between two hybrid programs. α^* is the iteration of α an arbitrary finite number of times, possibly zero.

The semantics is a possible world Kripke semantics where worlds represent the possible system states and the accessibility relation between worlds is reachability by hybrid transitions. The states are tuples of variables to which are assigned real values. The value of a variable x in a state ω is denoted by $[\![x]\!]_\omega$ and of a term θ in a state ω is denoted by $[\![\theta]\!]_\omega$.

The semantic of programs is given by the reachability relation $\rho_\nu(\alpha)$. It denotes the set of states ω reachable by the program α starting from the state ν.

Definition 2 (Reachability semantic of discrete programs)**.**

$$
\begin{aligned}
\rho_\nu(x := \theta) &= \{\omega \mid \omega = \nu \textit{ except that } [\![x]\!]_\omega = [\![\theta]\!]_\nu\} \\
\rho_\nu(?\varphi) &= \{\nu \mid \nu \models \varphi\} \\
\rho_\nu(\alpha;\beta) &= \textstyle\bigcup_{\omega \in \rho_\nu(\alpha)} \rho_\omega(\beta) \\
\rho_\nu(\alpha \cup \beta) &= \rho_\nu(\alpha) \cup \rho_\nu(\beta) \\
\rho_\nu(\alpha^*) &= \textstyle\bigcup_{n \in \mathbb{N}} \rho(\alpha^n) \textit{ with } \alpha^0 = ?\top \\
&\qquad \textit{and } \alpha^{n+1} = \alpha^n;\alpha
\end{aligned}
$$

where $\nu \models \varphi$ means that the state ν satisfies the formula φ.

We provide graphical examples of the reachability of each construct. The reachability set of the assignment from the state ν is all the states ω which are the same as ν, but for the value of x. For example, in the Figure 2.1, the value of x in the sate ω is now 42, but the value of y is unchanged.

The reachability set from the state ν of the test is the set formed of only the state ν, but which satisfies the formula φ. The values of the state are unchanged (cf Figure 2.26).

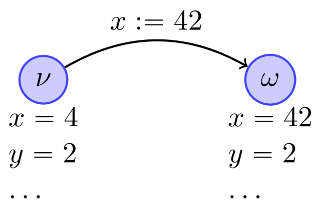


Figure 2.1: Assignment

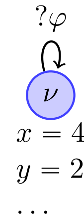


Figure 2.2: Test

A reachable state of $\alpha;\beta$ from a state ν is a state reachable by β from a reachable state of α. In the Figure 2.3, ν_3 is a reachable state from ν_2 by $y := y + 1$ and ν_2 is a reachable state of ν_1 by $x := 42$. Thus ν_3 is a reachable state from ν_1 by the program $x := 42; y := y + 1$.

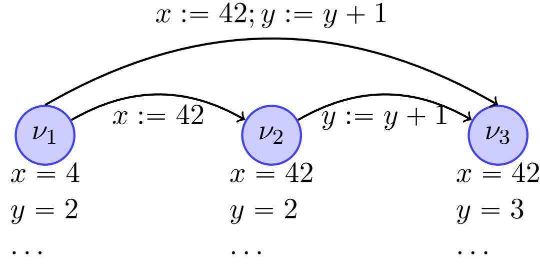


Figure 2.3: Sequence

A reachable state from the state ν by $\alpha \cup \beta$ is either a reachable state of α or of β. In the Figure 2.4, ω_1 is a reachable state from ν by $x := 42$ and ω_2 is a reachable state from ν by $y := y + 1$. Thus, ω_1 and ω_2 are reachable states of $x := 42 \cup y := y + 1$.

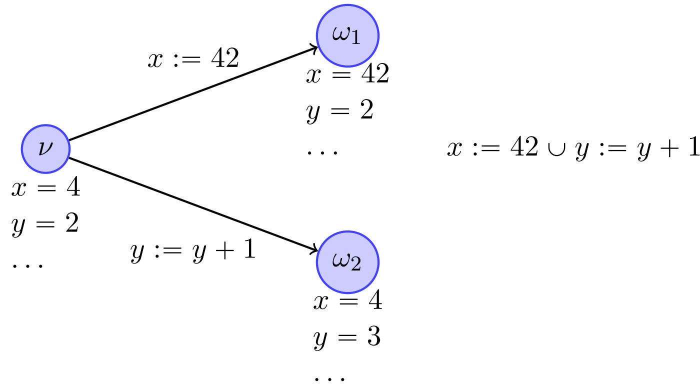


Figure 2.4: Non-deterministic choice

A state ω is reachable from the state ν by α^* if it is reachable by the program α^n which is n consecutive executions of α. For example, ν_2 is reachable by $y := y + 1$ and ν_3 by $y := y + 1; y := y + 1$. They are both reachable states of $(y := y + 1)^*$.

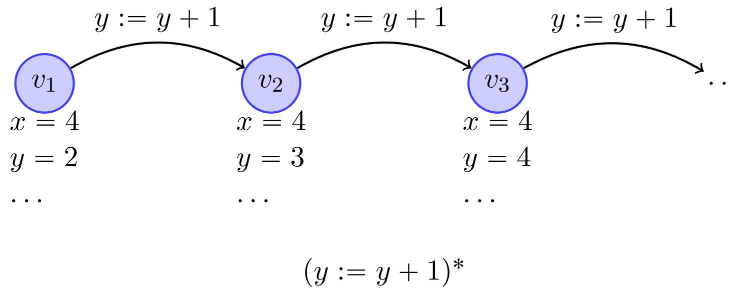


Figure 2.5: Iteration

We can define the program that computes the Fibonacci sequence.

Example 1 (Fibonacci sequence)**.**

$$\mathit{Fibonacci} \triangleq (F_n := F_{n+1}; F_{n+1} := F_{n+2}; F_{n+2} := F_{n+1} + F_n)^*$$

We assume that the formulas $F_n = 0$, $F_{n+1} = 1$ and $F_{n+2} = 1$ hold at the initialization.

We can also encode a simple speed-controller for a car.

Example 2 (Controller for a car)**.**

$$\big((?accel; a := A) \cup a := -b\big)^*$$

If the condition accel is true, the car may accelerate by A, but it can always brake by $-b$.

Encoding of usual programming structure We retrieve the `while` and `if-then-else` structure with the following encoding. The interested reader can find more encoding in [61, p. 167].

$\texttt{if}\ \varphi\ \texttt{then}\ \alpha\ \texttt{else}\ \beta \equiv (?\varphi; \alpha) \cup (?\neg\varphi; \beta)$
$\texttt{while}\ \varphi\ \texttt{do}\ \alpha \equiv (?\varphi; \alpha)^*; ?\neg\varphi$

2.1.2 Ordinary Differential Equations

In this section, we present the syntax and the semantic of Ordinary Differential Equations (ODEs) in $d\mathcal{L}$. We illustrate it with several examples.

The modeling of physical phenomena is mostly made by the use of differential equations since the foundational work of Leibniz and Newton. The idea is to consider any infinitesimal change in an equation to represent continuous motion. In mathematics, the change is made with respect to an arbitrary variable, but in physics, we are mostly interested by the change with respect to time. We adopt the convention that the variation of a quantity, represented by x, over the time is denoted by $\dot{x}$.

Definition 3 (Syntax of differential equation)**.**

$$\dot{X} = \Theta_X \ \ \& \ \ H$$

where H is a formula of $d\mathcal{L}$.

X is vector of variables $(x_1, \ldots, x_n)$ and Θ_X a vector $(\Theta_1, \ldots, \Theta_n)$ of terms of real arithmetic. The symbol $\&$ stands for a separator between the differential equation and the *evolution domain* H. It characterizes the domain in which it can evolve; if the formula H is not true, then the evolution is not considered. For example, the evolution domain of time, represented by the variable t, is $t \geqslant 0$ since it does not make sense to consider negative time by convention.

We assume that every differential equation has a solution which is unique. We define accordingly the semantic using this solution. We define the reachability for a differential equation with only one variable, but it is easily generalized for a vector of variables.

Definition 4 (Reachability semantics of differential equation)**.**

$$\begin{aligned}\rho_\nu(\dot{x} = \theta_x \ \ \& \ \ H) \ = \ \{f(r) \mid & \ \textit{there is a function } f : [0, r] \to \mathcal{S} \textit{ such that} \\ & f(t) \models \dot{x} = \theta \textit{ and } f(t) \models H \textit{ for every } t \in [0, r]\}\end{aligned}$$

$\mathcal{S}$ is the set of states. $f(t)$ has to be understood as a solution of the differential equation. The reachable states are every point to which the dynamic system can evolve providing that the evolution domain still holds.

For example, in the Figure 2.6, ω is a reachable state where the value of x is $f(2)$, but the value of y stays unchanged. Another reachable state is the one where the value of x is $f(2.1)$ and the value of y is unchanged. The behavior of an ODE can be understood as an assignment iterated, except that the iteration step is infinitesimal.

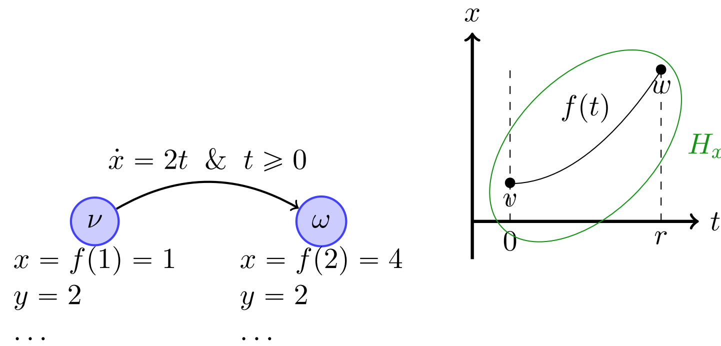


Figure 2.6: Differential equation

We can model most of physical systems of interest. A limit is the modeling of dynamic systems which do not admit solution, *e.g.* Partial Differential Equations (PDEs).

Example 3 (Passing of time)**.**

$$\dot{t} = 1 \;\; \& \;\; t \geqslant 0$$

t represents the time, and it evolves linearly. We consider also the time to be always positive by convention.

We are able to model more complex systems such as the speed of a car or the water-level in a tank or in several tanks.

Example 4 (Speed of a car)**.**

$$\dot{x} = v, \dot{v} = a \;\; \& \;\; v \geqslant 0$$

where x is the position of the car, v the velocity (speed) of the car and a the acceleration of the car.

It is a system of differential equations and we add in the evolution domain $v \geqslant 0$ because a speed is always considered positive.

Example 5 (Water-Tank)**.**

$$\dot{wl} = fin - fout \;\; \& \;\; wl \geqslant 0$$

where wl is the water-level, fin is the inlet flow and $fout$ is the outlet flow. We assume that the water-level is always positive, i.e. that it is not below the bottom of tank.

Example 6 (Water-Plant)**.** *We can represent the water-level of a network of connected water-tanks. For example, if the inlet flow of the second water-tank is the outlet flow of the first water-tank and similarly for the second and third water-tank.*

$$\begin{cases} wl_1 = fin_1 - fout_1 \\ wl_2 = fout_1 - fout_2 \\ wl_1 = fout_2 - fout_3 \end{cases} \quad \& \quad wl_1 \geqslant 0 \wedge wl_2 \geqslant 0 \wedge wl_3 \geqslant 0$$

We can model continuous systems and discrete systems, but the very interest of hybrid systems is the interaction of the two. The next section is devoted to several examples of such interactions.

2.1.3 Blending discrete and continuous aspects

We present in this subsection several examples of standard hybrid systems. We present the bouncing-ball example where the discrete interaction is not provoked by humans, and the class of computer-controlled systems which are designed by humans. The bouncing-ball example is a popular and simple hybrid system where the discrete interaction does not come from a controller.

Example 7 (Bouncing ball)**.** *It models the velocity v and height h of a falling ball. The discrete interaction comes from the bouncing on the floor. The ball falls until it touches the floor ($h = 0$), then it bounces back. The velocity of the bouncing back is function of the falling velocity moderated by a damping factor c accounting for loss of kinetic energy.*

$$(\dot{h} = v, \dot{v} = -g \;\;\&\;\; h \geqslant 0; ?h = 0; v := -cv)^*$$

g is the value of gravity. The differential equation evolves until the height h equals 0. Then, we can pass the test $?h = 0$ and we assign the value $-cv$ to the velocity which accounts for the bouncing. The iteration constructs allows to consider an arbitrary large number of such sequences.

Modeling Computer-Controlled System Computer-Controlled systems are frequently encountered and regroup systems where a continuous behavior, so-called *plant*, is monitored by a controller, in this case a computer with sensors and actuators. The controller is triggered by a timing condition. It executes at least every Δ units of time.

Example 8 (Time triggered)**.**

$$\begin{array}{lr} (((?accel; a := A) \cup a := -b); t_{Ctrl} := t; & \textit{(Control)} \\ \dot{x} = v, \dot{v} = a, \dot{t} = 1 \;\;\&\;\; v \geqslant 0 \wedge t \geqslant 0 \wedge t - t_{Ctrl} \leqslant \Delta)^* & \textit{(Plant)} \end{array}$$

We are now able to model hybrid systems in $d\mathcal{L}$, the next step is to express properties that such systems should satisfy, for example that the water-level of the tank is within some interval.

2.2 Expressing Properties in $d\mathcal{L}$

First Order Real Arithmetic is a first-order logic with terms of Real Arithmetic. It allows to represent polynomial terms, but not trigonometric or exponential terms. We present it in Subsection 2.2.1. Formulas in $d\mathcal{L}$ are First Order Real Arithmetic formulas augmented with the modality $[.]$. $[\alpha]\varphi$ means that the formula φ holds for every executions of the hybrid program α. We detail it in Subsection 2.2.2.

2.2.1 First-order Real Arithmetic

We define the syntax of *terms* of Real Arithmetic and their semantic. We define also the syntax of *formulas* of First-Order Real Arithmetic and their semantic. We present examples of properties that we can express and state the surprising result of Tarski about the decidability of the First Order Real Arithmetic.

Definition 5 (Terms of Real Arithmetic)**.**

$$\theta_1, \theta_2 ::= x \mid 0 \mid 1 \mid \theta_1 + \theta_2 \mid \theta_1 - \theta_2 \mid \theta_1 \times \theta_2 \mid \theta_1 \div \theta_2$$

The variables x are valuated in $\mathbb{R}$. We have the usual semantic for the operators $+,\ -,\ \times,\ \div$. It allows polynomials, *e.g.* $x + x^2 + x^3$, but does not allow more complex expressions like logarithms, exponential or trigonometric functions. The valuation of a term θ at a state ν is denoted by $[\![\theta]\!]_\nu$.

We define the formulas of Real Arithmetic by adding comparison operators between terms, $<, \leqslant, =, \geqslant, >$, and logical connectives.

Definition 6 (Formulas of First-Order Real Arithmetic)**.**

$$\begin{array}{rcl} \varphi, \psi & ::= & \theta_1 \sim \theta_2 \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \varphi \rightarrow \psi \mid \forall x \varphi \mid \exists x \varphi \mid \bot \\ & & \textit{where} \ \sim \in \{<, \leqslant, =, \geqslant, >\} \end{array}$$

We define the semantic only for the connectives $\sim$, $\bot$, $\wedge$, $\rightarrow$ and $\forall$. The others can be deduced from it.

Definition 7 (Semantic of formulas)**.** *The satisfiability of formulas is provided for any state* ν*:*

$$\begin{array}{lcl} \nu \models \theta_1 \sim \theta_2 & \Leftrightarrow & [\![\theta_1]\!]_\nu \sim [\![\theta_2]\!]_\nu, \ \sim \in \{<, \leqslant, =, \geqslant, >\} \\ \nu \models \varphi \wedge \psi & \Leftrightarrow & \nu \models \varphi \textit{ and } \nu \models \psi \\ \nu \models \varphi \rightarrow \psi & \Leftrightarrow & \textit{if } \nu \models \varphi \textit{ then } \nu \models \psi \\ \nu \models \forall x\, \varphi & \Leftrightarrow & \forall \omega \textit{ such that } \forall z, z \neq x \Rightarrow [\![z]\!]_\omega = [\![z]\!]_\nu \textit{ we have } \omega \models \varphi \\ \nu \not\models \bot & & \end{array}$$

We can express that a quantity x is within an interval $x \in [-42, 42]$ as $-42 \leqslant x \wedge x \leqslant 42$.

Example 9 (Water-level property)**.** *We require that the water-level wl in the water-tank stays between* 3 *and* 7 *:* $wl \geqslant 3 \wedge wl \leqslant 7$*. It means that there is always a sufficient outlet flow, and that it does not overflow.*

Tarski Algebra Surprisingly, contrary to the First-Order Natural Arithmetic, the First-Order Real Arithmetic is decidable, it is the so-called Tarski's Miracle [122]. The reason comes from the sign conservation between two zeros of a polynomial in $\mathbb{R}$, which is not the case for the natural arithmetic.

Theorem (Tarski's Miracle)**.** *The First-Order Real Arithmetic is decidable, with a complexity doubly exponential in time.*

Despite of the decidability result, it is still complex to prove formulas of Real Arithmetic. Much progress have been made these last decades with efficient SMT solvers like Z3 [39].

We want to ultimately prove that a system satisfies a property for all its executions, *i.e.* that it is an invariant. For example, we want that the water-level wl stays always between 3 and 7. We need modality to express such statement.

2.2.2 A modal logic

We present in this section the syntax and the semantic of the modal operator [.]. It allows to express safety properties of a system. We show how to encode the notion of contract. We present also the modal operator $\langle . \rangle$ to express liveness properties.

Modal operator [.] We add to the definition of formula of the Real Arithmetic the modality $[\alpha]\varphi$ which means that "for every reachable state by α, the formula φ is true".

Definition 8 (Formulas of $d\mathcal{L}$)**.**

$$\begin{array}{rcl} \varphi, \psi & ::= & [\alpha]\varphi \mid \theta_1 \sim \theta_2 \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \varphi \rightarrow \psi \mid \forall x \varphi \mid \exists x \varphi \mid \bot \\ & & \textit{where} \ \sim \in \{<, \leqslant, =, \geqslant, >\} \end{array}$$

where α is a hybrid program (Def. 1 and Def. 3). We retrieve the Real Arithmetic formulas defined in Def. 6.

The semantic is defined as follows.

Definition 9 (Semantic)**.**

$$\nu \models [\alpha]\varphi \Leftrightarrow \forall \omega \in \rho_\nu(\alpha), \ \omega \models \varphi$$

The modality [.] articulates the behavior of hybrid programs and formulas of Real Arithmetic.

Example 10. *The formula $[\dot{t} = 1, \dot{d} = e, \dot{e} = -d \ \ \& \ \ t \geqslant 0]d^2 + e^2 \leqslant r^2$ means that the property $d^2 + e^2 \leqslant r^2$ is always true for any execution of $\dot{t} = 1, \dot{d} = e, \dot{e} = -d$.*

Example 11 (Satisfaction of the water-level property)**.** *The formula $[Water - tank]3 \leqslant wl \leqslant 7$ means that the water-tank system satisfies the water-level property.*

Remark 1. *In $[\alpha]\varphi$, the formula φ is required to hold only at the end state of the execution. For example, the formula $[x := -1; x := 42]x \geqslant 0$ is valid although x is negative in the middle of the execution. An extension of $d\mathcal{L}$ with temporal constructs has been proposed to take into account the intermediary states [96]. However, most of systems of interest are modeled as a iterated hybrid program $\alpha^\star$. The semantic of the iteration implies that φ will be required to hold at every possible executions of α, including the initial state since it is allowed to have zero iteration.*

Assume-Guarantee reasoning The *Assume-Guarantee reasoning* is widely used to express properties under which a system works and the properties guaranteed. It consists in the specification of formulas which hold for the inputs, the *assumptions*, and for the outputs, the *guarantees*. A *contract* is the 2-tuple of assumptions and guarantees.

Definition 10 (Contract in $d\mathcal{L}$)**.** *Let α be a hybrid program, A the assumptions under which α behaves and G the guarantees provided. A* contract *is represented by the formula*

$$A \rightarrow [\alpha]G$$

Example 12. *We want to ensure that under the assumption that $F_n = 0 \wedge F_{n+1} = 1 \wedge F_{n+2} = 1$, the Fibonacci program in the Example 1 satisfies the property $F_{n+2} = F_{n+1} + F_n$. It is represented by the formula*

$$(F_n = 0 \wedge F_{n+1} = 1 \wedge F_{n+2} = 1) \rightarrow [\textit{Fibonacci}]F_{n+2} = F_{n+1} + F_n$$

Liveness properties The modality [.] allows to express *safety* properties, *i.e.* true for every execution of the system under consideration. We may be interested by *liveness* properties, *i.e.* properties which have to hold for at least one execution of the system under consideration. The notion of liveness in $d\mathcal{L}$ differs from the one in LTL or CTL. In the former, it is "there is one execution of the system for which the desired property holds", but for the latter, it is that "for every execution of a system, there is a possible execution which leads to a state where the property is true".

The liveness modality $\langle . \rangle$ is the dual of the safety modality.

Definition 11 (Syntax of $\langle . \rangle$).

$$\langle \alpha \rangle \varphi \triangleq \neg [\alpha] \neg \varphi$$

From the semantics of [.], we can deduce the semantics of $\langle . \rangle$.

Definition 12 (Semantics of $\langle . \rangle$).

$$\nu \models \langle \alpha \rangle \varphi \Leftrightarrow \exists \omega \in \rho_\nu(\alpha),\ \omega \models \varphi$$

Conclusion We have presented how to model hybrid systems in $d\mathcal{L}$ (Section 2.1) and the definition of formulas of $d\mathcal{L}$ (Section 2.2). We have also shown how to associate properties to hybrid programs. In order to verify that a hybrid program satisfies its properties, we need a proof system. The next section is devoted to the presentation of the rules of the sequent calculus of $d\mathcal{L}$.

2.3 Proving properties in $d\mathcal{L}$

This section is devoted to the detailed presentation of the proof system of $d\mathcal{L}$. We present briefly the general characteristics and definitions of sequent calculus in Subsection 2.3.1. In Subsection 2.3.2, we present proof rules to prove First-Order Real Arithmetic formulas. We present structural rules in Subsection 2.3.3. Theses rules are used to clarify proofs. In Subsection 2.3.4, we present specific rules to reason on the execution of hybrid programs.

2.3.1 Generalities on sequent calculus

We present in this section generalities on sequent calculus along with the definition of a sequent and how to read proof rules. We also state the definition of soundness and completeness for a proof system.

A *sequent calculus* is a deductive proof calculus which relies on proof rules. It is widely used to perform and represent proofs. Sequent calculus exhibits interesting aspects such as the fact that it is a syntactic approach and thus amenable to automation. It is also easy to implement proof strategies to augment its usability. Lastly, it is easily extendable to take into account other theories. Several theorem provers use sequents to represent proof reasoning, for example Coq [36], B-method [3], KeY [6].

It is characterized by the use of *sequents* $\Gamma \vdash \Delta$ where Γ (resp. Δ) is a finite set of formulas $\gamma_1, \ldots, \gamma_n$ (resp. $\delta_1, \ldots, \delta_m$). The meaning is "the conjunction of formulas in Γ implies the disjunction of formulas in Δ" and is thus equivalent to the formula $\bigwedge_{1 \leqslant i \leqslant n} \gamma_i \to \bigvee_{1 \leqslant j \leqslant m} \delta_j$.

$$\frac{\text{Premise } 1 \quad \dots \quad \text{Premise } n}{\text{Conclusion}}$$

Given a sequent, we can derive a *proof tree* by the use of proof rules. It is composed of n sequents above the bar, called *premises*, and one sequent under the bar, called the *conclusion*. The semantic is "If the premises are valid, the conclusion is valid".

The set of proof rules is the sequent calculus. An important property is the soundness of rules, *i.e.* that if the premises of a proof rule are valid, then the conclusion is valid.

Definition 13 (Soundness)**.** *A sequent calculus is* sound *if all its proof rules are sound. Equivalently, for any formula φ, if we have a derivation (denoted by $\vdash \varphi$), then the formula is valid (denoted by $\models \varphi$).*

It means that every formula proved with a sound sequent calculus is semantically valid. It links a syntactical and mechanical procedure with a semantic meaning. The converse is the completeness property. Every valid formula can be proved within the sequent calculus.

Definition 14 (Completeness)**.** *A sequent calculus is* complete *if for any valid formula φ, i.e. $\models \varphi$, we have a derivation, i.e. $\vdash \varphi$.*

A sequent calculus for a logical theory which exhibits both properties implies that the theory is decidable.

2.3.2 First Order Real Arithmetic

We present the rules to handle First-Order Real Arithmetic formulas. We define closing rules which are leaves of a proof tree, then propositional rules which are for logical connectives $\neg$, $\rightarrow$, $\wedge$ and $\vee$. We present quantifier rules for quantified formulas.

Closing rules *Closing rules* are rules which do not have any premises. They are used to close the proof tree, *i.e.* they are leaves. We present them in the Figure 2.7.

$$\frac{}{\Gamma, \varphi \vdash \varphi, \Delta}\text{ax} \qquad \frac{\text{Real Arithmetic}}{\Gamma \vdash \theta_1 \sim \theta_2, \Delta}\text{QE}$$

$$\frac{}{\Gamma, \bot \vdash \Delta}\bot \qquad \frac{}{\Gamma \vdash \top, \Delta}\top$$

Figure 2.7: Closing rules

The axiom rule ax means that "If φ is known and we try to prove φ, then the proof is valid". The rule $\bot$ is the *ex falso quod libet* reasoning; if we have false in the hypothesis, we can derive every formula, and thus our current goal. The rule $\top$ is the inverse. We try to prove that the goal $\top$ is true, which is the case by definition.

The rule QE is slightly different. It is the rule to handle the validity of terms of real arithmetic. They are are premises, but we do not consider them because we assume that there is an external procedure to handle them. In KeYmaera X, it is achieved by Z3. The name QE stands for Quantifier Elimination. A first-order theory admits *quantifier elimination*

if for each formula φ, a quantifier-free formula $\mathbf{qelim}(\varphi)$ can be effectively associated that is equivalent, *i.e.* $\varphi \longleftrightarrow \mathbf{qelim}(\varphi)$ and has no other free variables. Real Arithmetic enjoys such property.

Propositional rules *Propositional rules* are for the logical connectives $\rightarrow$, $\neg$, $\wedge$ and $\vee$. They are presented in the figures 2.8, 2.9, 2.10 and 2.11.

$$\frac{\Gamma, \varphi \vdash \psi, \Delta}{\Gamma \vdash \varphi \rightarrow \psi, \Delta} \rightarrow_r \qquad \frac{\Gamma \vdash \varphi, \Delta \qquad \Gamma, \psi \vdash \Delta}{\Gamma, \varphi \rightarrow \psi \vdash \Delta} \rightarrow_l$$

Figure 2.8: Rules $\rightarrow_r$ and $\rightarrow_l$

The rule $\rightarrow_r$ transcripts the semantic of a sequent. To prove the goal $\varphi \rightarrow \psi$ is the same as assuming φ and trying to prove the goal ψ. We have a dual rule for the case where the formula is in the hypothesis, *i.e.* on the left of a sequent. It splits in two premises. The left premise asks to prove φ and the right premise assumes ψ; to be able to use ψ as an hypothesis, we have to prove that φ is true.

$$\frac{\Gamma, \varphi \vdash \Delta}{\Gamma \vdash \neg\varphi, \Delta} \neg_r \qquad \frac{\Gamma \vdash \varphi, \Delta}{\Gamma, \neg\varphi \vdash \Delta} \neg_l$$

Figure 2.9: Rules $\neg_r$ and $\neg_l$

The rule $\neg_l$ means "To prove Δ knowing Γ and $\neg\varphi$, either Γ is sufficient or φ is true, thus $\neg\varphi$ is false". We have the dual rule $\neg_r$. We can derive them from the rules $\bot$ and $\rightarrow_l$:

$$\frac{\Gamma \vdash \varphi, \Delta}{\Gamma, \neg\varphi \vdash \Delta} \neg_l \quad \text{can be derived from} \quad \frac{\Gamma \vdash \varphi, \Delta \qquad \overline{\Gamma, \bot \vdash \Delta}\ \bot}{\Gamma, \varphi \rightarrow \bot \vdash \Delta} \rightarrow_l$$

The notion of *derived rule* is very important. It can be understood as "macros" for proof and it leads to proof programming [46]. It is one of the strengths of sequent calculus, and it is necessary to scale to complex systems.

$$\frac{\Gamma \vdash \varphi, \Delta \qquad \Gamma \vdash \psi, \Delta}{\Gamma \vdash \varphi \wedge \psi, \Delta} \wedge_r \qquad \frac{\Gamma, \varphi, \psi \vdash \Delta}{\Gamma, \varphi \wedge \psi \vdash \Delta} \wedge_l$$

Figure 2.10: Rules $\wedge_r$ and $\wedge_l$

The rule $\wedge_r$ has two premises; to prove the goal $\varphi \wedge \psi$, we have to prove the φ and ψ separately. The dual rule $\wedge_l$ transcripts the semantic of a sequent. The set of formulas on the left of $\vdash$ are interpreted as a conjunction.

$$\frac{\Gamma, \varphi \vdash \Delta \qquad \Gamma, \psi \vdash \Delta}{\Gamma, \varphi \vee \psi \vdash \Delta} \vee_l \qquad \frac{\Gamma \vdash \varphi, \psi, \Delta}{\Gamma \vdash \varphi \vee \psi, \Delta} \vee_r$$

Figure 2.11: Rules $\vee_r$ and $\vee_l$

Rules $\vee_r$ and $\vee_l$ are dual, and can be derived from $\wedge_r$ and $\wedge_l$ since we have $\varphi \vee \psi \leftrightarrow \neg(\neg\varphi \wedge \neg\psi)$.

Quantifier rules *Quantifier rules* handle formulas with a quantifier on topmost position.

$$\frac{\Gamma \vdash \varphi(x/x^0), \Delta}{\Gamma \vdash \forall x \varphi(x), \Delta} \forall_r \qquad \frac{\Gamma, \varphi(x/\theta) \vdash \Delta}{\Gamma, \forall x \varphi(x) \vdash \Delta} \forall_l$$

where x^0 is a fresh variable
and θ is a chosen term of Real Arithmetic

Figure 2.12: Rules $\forall_r$ and $\forall_l$

The notation $\varphi(x/x^0)$ means "the formula φ in which every occurrence of the variable x has been replaced by the variable x^0". It is a *substitution*. The interested reader can find more information in [94, p.65].

In the rule $\forall_r$, we replace every occurrences of x by a fresh variable x^0. In mathematics, to prove a universal property, we chose to reason with a parameter which has no connections with the rest of the formula. It is similar to the α-renaming in the λ-calculus. For the rule $\forall_l$, we replace x with a chosen term of real arithmetic. If we assume a universal property, we can instantiate it to use in a proof.

The existential quantifier rules are dual of the universal quantifier rules. They can be derived also from the equivalence $\exists x \varphi(x) \leftrightarrow \neg(\forall x, \neg\varphi(x))$.

$$\frac{\Gamma \vdash \varphi(x/\theta), \Delta}{\Gamma \vdash \exists x \varphi(x), \Delta} \exists_r \qquad \frac{\Gamma, \varphi(x/x^0) \vdash \Delta}{\Gamma, \exists x \varphi(x) \vdash \Delta} \exists_l$$

where x^0 is a fresh variable
and θ is a chosen term of Real Arithmetic

Figure 2.13: Existential rules

Invertible rules The rule $\forall_l$ is different from the previous rules; the proof engineer has to choose a term of real arithmetic. It is possible that the conclusion is provable, but that the premise is not. The proof tree 2.14 shows such possibility, where the instantiation of x has been made with a wrong variable z instead of choosing the variable y as in the proof tree 2.15.

$$\frac{z^2 \geqslant 0 \vdash y \geqslant 0}{\forall x . x^2 \geqslant 0 \vdash y \geqslant 0} \forall_l$$

Figure 2.14: Wrong choice of variable in rule $\forall_l$

Definition 15. *A rule is* invertible *if the conclusion is provable if and only if the premises are provable.*

$$\dfrac{y^2 \geqslant 0 \vdash y \geqslant 0}{\forall x. x^2 \geqslant 0 \vdash y \geqslant 0}\ \forall_l$$

Figure 2.15: Right choice of variable in rule $\forall_l$

Invertible rules play an important role into the automation of proof. They can be applied safely since they do not change the provability of the considered formula, and a possible heuristic is to favor their use. The propositional rules and the rule $\forall_r$ are invertible, but the rule $\forall_l$ is not. The rules presented in the following subsection are not invertible.

Theorem (Soundness theorem)**.** *The rules presented in figures 2.7 to 2.13 are sound.*

Proof. We detail the case of the rule $\wedge_r$. The others can be found in [98, p. 97]. We reason by induction on the structure of the formula. To prove that $\wedge_r$ is sound, we assume that each premise is valid. We prove then that the conclusion is valid.

Assume that $\models \varphi \vee \Delta$ and $\models \psi \vee \Delta$. We want to prove $\models (\varphi \wedge \psi) \vee \Delta$. We obtain it by application of the definition of the semantic of $\wedge$ (def. 7).

□

Conclusion The rules presented in figures 2.7 to 2.13 are sufficient to prove validity of First-Order Real Arithmetic formulas. For example, we can prove the validity of the formula $\big((p \to q) \wedge (r \to s) \wedge (p \vee r)\big) \to (q \vee s)$ (cf proof tree 2.16) or of the formula $x = 3 \to \forall x, x^2 \geqslant 0$ (cf proof tree 2.17).

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{r \to s, p \vdash p, q, s}\ \text{ax} \quad \dfrac{}{r \to s, q \vdash q, s}\ \text{ax}}{p \to q, r \to s, p \vdash q, s}\ \to_l \quad \dfrac{\text{Similar to left branch}}{p \to q, r \to s, r \vdash q, s}}{p \to q, r \to s, p \vee r \vdash q, s}\ \vee_l}{p \to q, r \to s, p \vee r \vdash (q \vee s)}\ \vee_r}{p \to q, (r \to s) \wedge (p \vee r) \vdash (q \vee s)}\ \wedge_l}{(p \to q) \wedge (r \to s) \wedge (p \vee r) \vdash (q \vee s)}\ \wedge_l}{\vdash \big((p \to q) \wedge (r \to s) \wedge (p \vee r)\big) \to (q \vee s)}\ \to_r$$

Figure 2.16: Proof tree of the formula $\big((p \to q) \wedge (r \to s) \wedge (p \vee r)\big) \to (q \vee s)$

For the proof of $\big((p \to q) \wedge (r \to s) \wedge (p \vee r)\big) \to (q \vee s)$, we apply mechanically the proof rules and there is no need for a human intervention. This proof may be perform automatically.

$$\dfrac{\dfrac{\dfrac{\text{Real Arithmetic}}{x = 3 \vdash x_0^2 \geqslant 0}\ \text{QE}}{x = 3 \vdash \forall x. x^2 \geqslant 0}\ \forall_r}{\vdash x = 3 \to \forall x. x^2 \geqslant 0}\ \to_r$$

Figure 2.17: Proof tree of the formula $x = 3 \to \forall x. x^2 \geqslant 0$

For the proof of $x = 3 \rightarrow \forall x.x^2 \geqslant 0$, we apply the logical rule $\rightarrow_r$ and $\forall_r$ to obtain a Real Arithmetic term $x_0^2 \geqslant 0$ to prove. It is a well-known result that the square of a value is always positive. It is thus closed by the rule QE.

2.3.3 Structural rules

We present in this subsection structural rules *weakening* and *cut.* they do not change the power of the proof system; a provable formula is still provable without theses rules. But they are important to structure a proof, hence for the scalability of proof of complex systems. We present how we can derive the *modus ponens* rule with the cut rule.

Weakening rules In the left branch of the proof tree 2.16, the formulas $r \rightarrow s$ in the hypothesis and s in the goal are not used in the proof, and diminish the readability of a proof. We introduce the *weakening* rules to handle this problem.

$$\frac{\Gamma \vdash \Delta}{\Gamma \vdash \varphi, \Delta}\,\mathrm{W}_r \qquad\qquad \frac{\Gamma \vdash \Delta}{\Gamma, \varphi \vdash \Delta}\,\mathrm{W}_l$$

Figure 2.18: Rules W_r and W_l

The rule W_l allows to discard a formula in hypothesis. It is not invertible. The sequent in the conclusion can be valid, but the resulting sequent in the premise may not be. Nevertheless, the rule is still sound since, if we are able to prove a goal with a set of hypothesis Γ, we are able to prove it with one additional hypothesis. The rule W_r is the dual.

Weakening rules are intended to improve readability, but are also precious allies in the conduction of proof in an interactive theorem prover. By discarding useless formulas, we narrow the proof search and help the computer to automatically prove a goal.

Cut rule The *Cut* rule allows to introduce new formulas in a proof to strengthen hypotheses. It is the transcript of the use of lemmas in a standard mathematical proof.

$$\frac{\Gamma, \varphi \vdash \Delta \qquad \Gamma \vdash \varphi, \Delta}{\Gamma \vdash \Delta}\,\mathrm{Cut}$$

Figure 2.19: Cut rule

Examples Let us assume that we want to prove the validity of the formula $x \geqslant 0 \rightarrow x+y^2 \geqslant 0$, and we already have a proof Π of the formula $\forall x, x^2 \geqslant 0$ (cf proof tree 2.17). We cut this formula in the proof tree to achieve our proof tree 2.20.

It is a very important rule for the structuring of proofs. It allows to focus on one part of the problem. The rule Cut is also very useful to define derived rules.

The *modus ponens* rule By using the Cut rule, we can define a specific proof rule for the *Modus Ponens* (MP) reasoning (cf Fig. 2.21).

If we know $\varphi \rightarrow \psi$ and φ, then we can deduce ψ. The proof rule derives from the proof tree 2.22.

$$\dfrac{\dfrac{\dfrac{\dfrac{\text{Real Arithmetic}}{y^2 \geqslant 0, x \geqslant 0 \vdash x + y^2 \geqslant 0}\,\text{QE}}{\forall x.x^2, x \geqslant 0 \vdash x + y^2 \geqslant 0}\,\forall_l \qquad \dfrac{\dfrac{\Pi}{\vdash \forall x.x^2 \geqslant 0}}{x \geqslant 0 \vdash \forall x.x^2 \geqslant 0, x + y^2 \geqslant 0}\,\text{W}_l, \text{W}_r}{x \geqslant 0 \vdash x + y^2 \geqslant 0}\,\text{Cut}}{\vdash x \geqslant 0 \rightarrow x + y^2 \geqslant 0}\,\rightarrow_r$$

Figure 2.20: Proof tree of $x \geqslant 0 \rightarrow x + y^2 \geqslant 0$

$$\dfrac{\Gamma \vdash \varphi \rightarrow \psi, \Delta \qquad \Gamma \vdash \varphi, \Delta}{\Gamma \vdash \psi, \Delta}\,\text{MP}$$

Figure 2.21: *Modus Ponens* rule.

$$\dfrac{\dfrac{\dfrac{}{\Gamma, \varphi \vdash \varphi, \psi, \Delta}\,\text{ax} \qquad \dfrac{}{\Gamma, \psi \vdash \psi, \Delta}\,\text{ax}}{\Gamma, \varphi \rightarrow \psi, \varphi \vdash \psi, \Delta} \qquad \dfrac{\dfrac{\Gamma \vdash \varphi \rightarrow \psi, \Delta}{\Gamma, \varphi \vdash \varphi \rightarrow \psi, \Delta}\,\text{W}_l}{\Gamma, \varphi \vdash \varphi \rightarrow \psi, \psi, \Delta}\,\text{W}_r}{\Gamma, \varphi \vdash \psi, \Delta}\,\text{Cut} \qquad \dfrac{\Gamma \vdash \varphi, \Delta}{\Gamma \vdash \varphi, \psi, \Delta}\,\text{W}_r}{\Gamma \vdash \psi, \Delta}\,\text{Cut}$$

Figure 2.22: Proof tree for the derivation of the *Modus Ponens* rule

We have covered the proof rules for the standard logical connectives. The next step is to consider proof rules to reason about hybrid programs.

2.3.4 Hybrid program rules

This subsection regroups the rules used to reason on runs of a hybrid program. We recall that $[\alpha]\varphi$ means that "the formula φ holds for every run of the hybrid program α".

Assignment The assignment rule [:=] performs the change of value in the formula φ by replacing every occurrence of x in φ by the term θ.

$$\dfrac{\Gamma \vdash \varphi(\theta), \Delta}{\Gamma \vdash [x := \theta]\varphi(x), \Delta}\,[:=]$$

Figure 2.23: [:=] rule

We have to be careful with the phenomenon of *capture of variables* as in the β-reduction in λ-calculus. We need a bound variable renaming process similar to α-conversion. Intuitively, bound variables are variables that can change over the execution of a given system.

Bound variable renaming

Definition 16 (Bound variable)**.** *A variable x is* bound *if it is under the scope of a quantifier, e.g. $\forall x \varphi(x)$ or $\exists x \varphi(x)$, or if it is in the scope of a modality where x is the variable assigned,*

e.g. $x := \theta$*, or the variable derived, e.g.* $\dot{x} = \theta$*.*

Bound variables of a hybrid program α are variables that depend of the execution of α. The renaming of bound variables is necessary to ensure soundness. For example, we have the two proof trees 2.24 and 2.25, one without taking care of renaming bound variables, the other one with renaming. The first one is obviously unsound and does not match the semantic of the assignment.

$$\dfrac{\dfrac{x > 0, z > 0 \vdash z + z > z + z}{x > 0, z > 0 \vdash [z := z + z]z > z}\,[:=]}{x > 0, z > 0 \vdash [x := z][z := z + x]z > x}\,[:=]$$

Figure 2.24: No bound variable renaming

In Figure 2.24, the value of the variable z assigned to x is not the same as the variable z in the second modality. It leads to a proof stuck, and there is thus no means to prove a formula clearly valid.

$$\dfrac{\dfrac{\dfrac{\dfrac{\text{Real Arithmetic}}{x > 0, z > 0 \vdash z + z > z}\,\text{QE}}{x > 0, z > 0 \vdash [u := z + z]u > z}\,[:=]}{x > 0, z > 0 \vdash [x := z][u := z + x]u > x}\,[:=]}{x > 0, z > 0 \vdash [x := z][z := z + x]z > x}\,\text{Renaming } z \rightsquigarrow u$$

Figure 2.25: With bound variable renaming

In Figure 2.25, we rename the bound variable z in the second modality by a fresh variable u to avoid confusion. The renaming step will not be made explicit in the next sections, but is performed “on the fly”.

Test, sequence and non-deterministic choice The rule [?] is equivalent to try to prove ψ under the assumption φ. It amounts to add φ as an hypothesis for the proof.

$$\dfrac{\Gamma \vdash \varphi \rightarrow \psi, \Delta}{\Gamma \vdash [?\varphi]\psi, \Delta}\,[?]$$

Figure 2.26: Rules for the test

The rule [;] dissociates the two programs α and β.

$$\dfrac{\Gamma \vdash [\alpha][\beta]\varphi, \Delta}{\Gamma \vdash [\alpha;\beta]\varphi, \Delta}\,[;]$$

Figure 2.27: Rules for the sequence

The rule $[\cup]$ creates two premises, one for the execution of α and one for the execution of β. We have to prove that φ holds for each possibility.

$$\frac{\Gamma \vdash [\alpha]\varphi, \Delta \qquad \Gamma \vdash [\beta]\varphi, \Delta}{\Gamma \vdash [\alpha \cup \beta]\varphi, \Delta}\,[\cup]$$

Figure 2.28: Rules for the non-deterministic assignment

In Figure 2.29, we present an example of their use. It is a simple `if-then-else` program.

$$\frac{\dfrac{\dfrac{y > 0 \vdash x \geqslant 0 \rightarrow [z := x + y^2]z \geqslant 0}{y > 0 \vdash [?x \geqslant 0][z := x + y^2]z \geqslant 0}\,[?]}{y > 0 \vdash [?x \geqslant 0; z := x + y^2]z \geqslant 0}\,[;] \qquad \dfrac{\dfrac{y > 0 \vdash x < 0 \rightarrow [z := x + y^2]z \geqslant 0}{y > 0 \vdash [?x < 0][z := x^2 + y]z \geqslant 0}\,[?]}{y > 0 \vdash [?x < 0; z := x^2 + y]z \geqslant 0}\,[;]}{y > 0 \vdash [(?x \geqslant 0; z := x + y^2) \cup (?x < 0; z := x^2 + y)]z \geqslant 0}\,[\cup]$$

Figure 2.29: Example of use of rules $[?]$, $[;]$ and $[\cup]$

We can prove that for every execution of the following `if-then-else` program : $?\varphi; x := 0 \cup ?\neg\varphi; x := 1$, the formula $\varphi \rightarrow x = 0 \wedge \neg\varphi \rightarrow x = 1$ holds. The proof is in the proof tree 2.30.

$$\frac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{Real Arithmetic}}{\varphi, \varphi \vdash 0 = 0}\,\text{QE} \qquad \dfrac{\dfrac{}{\bot \vdash 0 = 1}\,\bot}{\neg\varphi, \varphi \vdash 0 = 1}}{\varphi \vdash \varphi \rightarrow 0 = 0 \wedge \neg\varphi \rightarrow 0 = 1}\,\wedge_r, \rightarrow_r}{\varphi \vdash [x := 0]\varphi \rightarrow x = 0 \wedge \neg\varphi \rightarrow x = 1}\,[:=]}{\vdash [?\varphi][x := 0]\varphi \rightarrow x = 0 \wedge \neg\varphi \rightarrow x = 1}\,[?], \rightarrow_r}{\vdash [?\varphi; x := 0]\varphi \rightarrow x = 0 \wedge \neg\varphi \rightarrow x = 1}\,[;] \qquad \Pi}{\vdash [?\varphi; x := 0 \cup ?\neg\varphi; x := 1]\varphi \rightarrow x = 0 \wedge \neg\varphi \rightarrow x = 1}\,[\cup]$$

Figure 2.30: Proof tree of $[?\varphi; x := 0 \cup ?\neg\varphi; x := 1]\varphi \rightarrow x = 0 \wedge \neg\varphi \rightarrow x = 1$

The branch Π in the proof tree 2.30 is, *mutatis mutandis*, developed as the left branch already detailed.

Remark 2. *The rules* $[:=]$, $[?]$, $[;]$ *and* $[\cup]$ *decompose the system without any choice involved. They are syntactically directed and thus amenable to automation.*

Iteration The rule [Ind] for the iteration is presented in the Figure 2.31. The reasoning is standard induction. We have two premises, one for the initial step and one for the induction step. The notation $\forall^\alpha\varphi$ is the skolemization of the formula φ by the bound variables of the hybrid program α.

$$\dfrac{\overbrace{\Gamma \vdash \varphi, \Delta}^{\text{Initial step}} \quad \overbrace{\Gamma \vdash \forall^{\alpha}(\varphi \rightarrow [\alpha]\varphi), \Delta}^{\text{Induction step}}}{\Gamma \vdash [\alpha^*]\varphi, \Delta} \text{[Ind]}$$

Figure 2.31: Rule [Ind]

To prove that a formula φ holds for every run of $\alpha^\star$, we have to prove that it holds for zero iteration, the *initial step*, and that if it holds an arbitrary number n of iterations, then it holds for the next iteration, the *induction step*.

The skolemization of a formula φ by a set of variables $\{x_1, \ldots, x_n\}$ is the addition of universal quantification over these variables in front of φ, *i.e.* $\forall x_1, \ldots \forall x_n \varphi$. It is a necessary abstraction to ensure soundness. If we do not add it, we can prove obviously false formulas as for example $x = 1 \rightarrow [(x := x-1)^*]x \geqslant 0$ (proof tree 2.32).

$$\dfrac{\dfrac{\dfrac{\text{Real Arithmetic}}{x = 1 \vdash x \geqslant 0}\text{QE} \quad \dfrac{\dfrac{\dfrac{\dfrac{\text{Real Arithmetic}}{x = 1 \vdash x - 1 \geqslant 0}\text{QE}}{x = 1, x \geqslant 0 \vdash x - 1 \geqslant 0}W_l}{x = 1, x \geqslant 0 \vdash [x := x-1]x \geqslant 0}[:=]}{x = 1 \vdash x \geqslant 0 \rightarrow [x := x-1]x \geqslant 0}\rightarrow_r}{x = 1 \vdash [(x := x-1)^*]x \geqslant 0}\text{[Ind]}}{\vdash x = 1 \rightarrow [(x := x-1)^*]x \geqslant 0}\rightarrow_r$$

Figure 2.32: Incorrect proof without skolemisation

In the Figure 2.32, the problem comes from that we can use the formula $x = 1$ to conclude the induction step, but it is the initial value of x.

Fibonacci We can prove that our implementation of Fibonacci (Example 1) satisfies the property $F_{n+2} = F_{n+1} + F_n$. We use the following notations:

- Fibonacci $\triangleq (body)^*$
- where $body \triangleq F_n := F_{n+1}; F_{n+1} := F_{n+2}; F_{n+2} := F_{n+1} + F_n$
- $Init \triangleq F_n = 0, F_{n+1} = 1, F_{n+2} = 1$
- $\varphi \triangleq F_{n+2} = F_{n+1} + F_n$

Our goal is to exhibit a proof tree of the sequent $Init \vdash$ [Fibonacci]φ.

$$\dfrac{\dfrac{\Pi_{init}}{Init \vdash \varphi} \quad \dfrac{\Pi_{Step}}{Init \vdash \forall F_n, F_{n+1}, F_{n+2}(\varphi \rightarrow [body]\varphi)}}{Init \vdash [(body)^*]\varphi}\text{[Ind]}$$

Figure 2.33: Proof tree for Fibonacci

We apply the induction rule [Ind] and it results in two premises: Π_{Init} and Π_{Step}. We handle them separately in their respective branches 2.34 and 2.35.

$$\dfrac{\dfrac{\dfrac{\text{Real Arithmetic}}{F_n = 0, F_{n+1} = 1, F_{n+2} = 1 \vdash 1 = 1 + 0}\text{QE}}{F_n = 0, F_{n+1} = 1, F_{n+2} = 1 \vdash F_{n+2} = F_{n+1} + F_n}\text{Evaluation of } F_n,\ F_{n+1} \text{ and } F_{n+2}}{Init \vdash \varphi}$$

Figure 2.34: Branch Π_{Init}

For the initial step (branch Π_{init}), we verify that φ holds initially, *i.e.* if it is true after zero execution of the system. We just have to replace the variables F_n, F_{n+1} and F_{n+2} in the right-hand side by their values in the left-hand side. We can conclude by trivial arithmetic.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash F^0_{n+2} + F^0_{n+1} = F^0_{n+2} + F^0_{n+1}}\text{ax}}{\vdash [F^1_{n+2} := F^0_{n+2} + F^0_{n+1}]F^1_{n+2} = F^0_{n+2} + F^0_{n+1}}[;],[:=]}{\vdash [F^1_{n+1} := F^0_{n+2}; F^1_{n+2} := F^1_{n+1} + F^0_{n+1}]F^1_{n+2} = F^1_{n+1} + F^0_{n+1}}[;],[:=]}{\vdash [F^0_n := F^0_{n+1}][F^1_{n+1} := F^0_{n+2}; F^1_{n+2} := F^1_{n+1} + F^0_n]\varphi^1}[:=]}{\varphi^0 \vdash [F^0_n := F^0_{n+1}; F^1_{n+1} := F^0_{n+2}; F^1_{n+2} := F^1_{n+1} + F^0_n]\varphi^1}\text{W}_l,\ [;]}{\varphi^0 \vdash [F^0_n := F^0_{n+1}; F^0_{n+1} := F^0_{n+2}; F^0_{n+2} := F^0_{n+1} + F^0_n]\varphi^0}\text{Renaming}}{Init \vdash \varphi^0 \to [body^0]\varphi^0}\text{W}_l,\ \to_r}{Init \vdash \forall F_n, F_{n+1}, F_{n+2}(\varphi \to [body]\varphi)}\forall_r$$

Figure 2.35: Branch Π_{Step}

For the induction step (branch Π_{Step}), we apply the rule $\forall_r$, and thus replace every occurrence of variables F_n, F_{n+1} and F_{n+2} by F^0_n, F^0_{n+1} and F^0_{n+2} which are assumed fresh. $body^0$ (resp. φ^0) is the program *body* (resp. formula φ) where this replacement has been made. We discard then the hypothesis $Init$ which brings no information on the new variables and pass φ^0 in hypothesis by the use of the rule $\to_r$. We rename the variables to avoid confusion for the multiple applications of rule [:=].

We deconstruct then the body with the rules [;] and [:=] until we obtain the formula $F^0_{n+2} + F^0_{n+1} = F^0_{n+2} + F^0_{n+1}$. We conclude by the rule *ax*.

Generalization The rule [Gen] allows to replace the invariant ψ by another invariant φ if we prove that φ is stronger that ψ.

$$\dfrac{\Gamma \vdash [\alpha]\varphi, \Delta \qquad \Gamma \vdash \forall^\alpha(\varphi \to \psi), \Delta}{\Gamma \vdash [\alpha]\psi, \Delta}[\text{Gen}]$$

Figure 2.36: [Gen] rule

The skolemization in the right premise of the rule presented in Figure 2.36 is for soundness purposes as the rule in [Ind] (cf Fig. 2.31).

Induction rule with a stronger invariant There is an other version of the induction rule where a stronger invariant is introduced, the rule [Ind2]. We present it in the Figure 2.37.

$$\frac{\Gamma \vdash \varphi, \Delta \qquad \Gamma \vdash \forall^{\alpha}(\varphi \to [\alpha]\varphi), \Delta \qquad \Gamma \vdash \forall^{\alpha}(\varphi \to \psi), \Delta}{\Gamma \vdash [\alpha^{*}]\psi, \Delta} \text{[Ind2]}$$

Figure 2.37: Rule [Ind2]

We introduce a stronger invariant φ instead of the previous invariant ψ. We have to prove the initial step and the induction step, but also to prove that φ implies ψ.

It can be obtained as a derived rule by the combination of rules [Gen] and [Ind] as shown in the Figure 2.38.

$$\frac{\dfrac{\Gamma \vdash \varphi, \Delta \qquad \Gamma \vdash \forall^{\alpha}(\varphi \to [\alpha]\varphi), \Delta}{\Gamma \vdash [\alpha^{*}]\varphi, \Delta} \text{[Ind]} \qquad \Gamma \vdash \forall^{\alpha}(\varphi \to \psi), \Delta}{\Gamma \vdash [\alpha^{*}]\psi, \Delta} \text{[Gen]}$$

Figure 2.38: Derivation of rule [Ind2]

Remark 3. *The main difficulty of the induction rule is the finding of a suitable invariant. There exists some heuristics to guess invariant, but no complete algorithms.*

ODESolve rule The rule [ODESolve] asks to provide a solution of the ODE to prove that φ is invariant.

$$\frac{\Gamma \vdash \forall t \geqslant 0\big((\forall 0 \leqslant s \leqslant t, [x := sol(s)]H) \to [x := sol(t)]\varphi\big), \Delta}{\Gamma \vdash [\dot{x} = f(x) \ \& \ H]\varphi, \Delta} \text{[ODEsolve]}$$

Figure 2.39: Rule [ODEsolve]

In the rule 2.39, the formula $sol(t)$ is the solution of the ODE $\dot{x} = f(x)$. We replace the occurrences of x by its solution in the evolution domain H and in the invariant φ. The skolemization is for soundness. t represents the evolution of time. The left part of the implication corresponds to the assumption that the evolution domain holds.

Example 13. *We present in the Figure 2.40 an example of the use of the rule [ODESolve]. Consider the water-level ODE in the Example 5. The solution of the water-level ODE is* $wl = (fin - fout)t + 4$*. We restrict its temporal evolution to one second by adding the formula* $0 \leqslant t \leqslant 1$ *in the evolution domain. Init denotes the initial value of the water-level,* $wl = 4$ *and the value of the parameters* $fin = 0$ *(i.e. the inlet valve is closed) and* $fout = 0.75$*.*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{Real Arithmetic}}{Init, \forall s\ 0 \leqslant s \leqslant 1 \vdash (fin - fout)t + 4 \leqslant 7}\ \text{QE}}{Init \vdash (\forall s\ 0 \leqslant s \leqslant 1) \rightarrow (fin - fout)t + 4 \leqslant 7}\ \rightarrow_r}{Init \vdash \forall t \geqslant 0\big((\forall s\ 0 \leqslant s \leqslant 1) \rightarrow (fin - fout)t + 4 \leqslant 7\big)}\ \forall_r}{Init \vdash \forall t \geqslant 0\big((\forall s[t := s]0 \leqslant t \leqslant 1) \rightarrow [wl := (fin - fout)t + 4]wl \leqslant 7\big)}\ [:=], [:=]}{Init \vdash [\dot{wl} = fin - fout\ \ \&\ \ 0 \leqslant t \leqslant 1]wl \leqslant 7}\ [\text{ODESolve}]$$

Figure 2.40: Example of use of the rule [ODESolve]

We apply the rule [ODESolve], then the assignment rule [:=] *twice to replace the variables by their solution. We deconstruct the formula* $\forall t \geqslant 0\big((\forall s\ 0 \leqslant s \leqslant 1) \rightarrow (fin - fout)t + 4 \leqslant 7\big)$*, and then discharge it to an external solver, e.g. Z3.*

Structural differential rules The Differential Cut rule [DC] and the Differential Weakening rule [DW] are two structural rules for the evolution domain. They are analogous to the rule Cut and the weakening rules W_l and W_r in Subsection 2.3.3.

The rule [DW] in Figure 2.41 means that the evolution domain is sufficiently strong to prove the guarantee without considering the behavior.

$$\dfrac{\Gamma \vdash \forall x(H \rightarrow \varphi), \Delta}{\Gamma \vdash [\dot{x} = f(x)\ \ \&\ \ H]\varphi, \Delta}\ [\text{DW}]$$

Figure 2.41: Rule [DW]

We have an example of its use in the Figure 2.42.

$$\dfrac{\dfrac{\text{Real Arithmetic}}{Init \vdash \forall wl, t\big((0 \leqslant t \leqslant 1 \wedge wl \leqslant 7 \wedge 3 \leqslant wl) \rightarrow 3 \leqslant wl \leqslant 7\big)}\ \text{QE}}{Init \vdash [\dot{wl} = fin - fout, \dot{t} = 1\ \ \&\ \ 0 \leqslant t \leqslant 1 \wedge wl \leqslant 7 \wedge 3 \leqslant wl]3 \leqslant wl \leqslant 7}\ [\text{DW}]$$

Figure 2.42: Example of use of the rule [DW]

We need to have a strong evolution domain to apply the rule [DW]. But it is usually not the case when we model a hybrid system. It is typically reinforced by multiple application of the rule [DC].

The rule [DC] in the Figure 2.43 introduces a new formula ψ in the evolution domain (right premise) provided we prove that the formula ψ is an invariant (left premise).

$$\dfrac{\Gamma \vdash [\dot{x} = f(x)\ \ \&\ \ H]\psi, \Delta \qquad \Gamma \vdash [\dot{x} = f(x)\ \ \&\ \ H \wedge \psi]\varphi, \Delta}{\Gamma \vdash [\dot{x} = f(x)\ \ \&\ \ H]\varphi, \Delta}\ \text{DC}$$

Figure 2.43: Rule [DC]

As for the rule Cut (Figure 2.19), it is very useful for the structuration of proofs. We illustrate it in the Figure 2.44.

Example 14 (Water-level example)**.** *In the water-level example (cf Ex. 5 and 13), we want now prove that the water-level does not overflow ($wl \leqslant 7$)* and *that it stays above a limit ($3 \leqslant wl$) in order to keep a sufficient pressure on the outlet valve. We want a proof of the sequent $Init \vdash [\dot{wl} = fin - fout \ \& \ 0 \leqslant t \leqslant 1]3 \leqslant wl \leqslant 7$. With the help of the rule [DC], we can consider each case separately.*

In the Figure 2.44, we first apply the rule [DC] twice to isolate each case. Finally, we apply the rule [DW] since the evolution contain the formula $3 \leqslant wl \leqslant 7$*. The branch* Π_1 *is already considered in the Figure 2.40. Mutatis mutandis, the branch* Π_2 *is closed by the same reasoning. The branch* Π_3 *is trivially closed .*

$$\dfrac{\dfrac{\Pi_3}{Init \vdash \forall wl\big((0 \leqslant t \leqslant 1 \wedge wl \leqslant 7 \wedge 3 \leqslant wl) \rightarrow 3 \leqslant wl \leqslant 7\big)}}{Init \vdash [\dot{wl} = fin - fout \ \& \ 0 \leqslant t \leqslant 1 \wedge wl \leqslant 7 \wedge 3 \leqslant wl]3 \leqslant wl \leqslant 7} \text{[DW]}$$

$$\dfrac{\dfrac{\Pi_2}{Init \vdash [\dot{wl} = fin - fout \ \& \ 0 \leqslant t \leqslant 1 \wedge wl \leqslant 7]3 \leqslant wl}}{Init \vdash [\dot{wl} = fin - fout \ \& \ 0 \leqslant t \leqslant 1 \wedge wl \leqslant 7]3 \leqslant wl \leqslant 7} \text{[DC]}$$

$$\dfrac{\dfrac{\Pi_1}{Init \vdash [\dot{wl} = fin - fout \ \& \ 0 \leqslant t \leqslant 1]wl \leqslant 7}}{Init \vdash [\dot{wl} = fin - fout \ \& \ 0 \leqslant t \leqslant 1]3 \leqslant wl \leqslant 7} \text{[DC]}$$

Figure 2.44: Example of use of rule [DC]

Differential induction The *differential induction* rule [DI] is similar to the induction rule, but for differential equation. In the case of discrete iteration, we are interested in the change between one step, the induction step. The step in a differential equation is just infinitesimal and corresponds to the derivation of the equation. The rule is presented in the Figure 2.45.

$$\dfrac{\overbrace{\Gamma \vdash \varphi, \Delta}^{\text{Initial step}} \qquad \overbrace{\Gamma \vdash \forall^{\alpha}(H \rightarrow \varphi'^{\theta}_{\dot{x}}), \Delta}^{\text{Differential Induction step}}}{\Gamma \vdash [\dot{x} = \theta \ \& \ H]\varphi, \Delta} \text{[DI]}$$

where $\varphi'^{\theta}_{\dot{x}}$ means that we substitute every occurrence of $\dot{x}$ by θ
and φ' is the derivation of φ as in def. 17 and 18

Figure 2.45: Differential induction rule

The left premise is the initial step. It demands that φ holds at the initial time. In the right premise, the differential induction step, we derive the formula φ according to Definitions 17 and 18 which implement the idea of differential step. We replace then the occurrences of $\dot{x}$ by θ.

Definition 17 (Derivation of terms)**.**

$$\begin{array}{lcll} (r)' & = & 0 & \textit{for } r \in \mathbb{Q} \\ (x)' & = & \dot{x} & \textit{for variable } x \\ (a+b)' & = & (a)' + (b)' & \\ (a-b)' & = & (a)' - (b)' & \\ (a \cdot b)' & = & (a)' \cdot b + a \cdot (b)' & \\ (a/b)' & = & \frac{(a)' \cdot b - a \cdot (b)'}{b^2} & \end{array}$$

They are the usual rules of the derivation calculus as taught in high school.

Example 15. *If $\theta \triangleq x^2 + y^2$, then $\theta' \triangleq 2x\dot{x} + 2y\dot{y}$*

We define also the derivation of formulas.

Definition 18 (Derivation of formulas)**.**

$$\begin{array}{lcll} (\theta_1 \sim \theta_2)' & = & (\theta_1)' \sim (\theta_2)' & \textit{except if } \sim \textit{ is } \neq \\ (F \wedge G)' & = & (F)' \wedge (G)' & \\ (F \vee G)' & = & (F)' \wedge (G)' & \\ (\forall x F)' & = & \forall x (F)' & \\ (\exists x F)' & = & \forall x (F)' & \end{array}$$

The rule [DI] allows to reason on differential equations without having to provide a solution.

Example 16. *Let us consider the differential equations $\dot{d} = e, \dot{e} = -d$ which represent an object moving along a circle of radius r in the plane with coordinates d and e. The solution involves trigonometric functions, and therefore is very difficult to handle. We want to prove that $d^2 + e^2 = r^2$ holds at any moment in the system. We apply the differential induction rule and then the several rules of derivation until the obtaining of a First-Order formula of the Real Arithmetic.*

$$\dfrac{\dfrac{\text{Real Arithmetic}}{d = 0, e = 1, r = 1 \vdash d^2 + e^2 = r^2}\,\text{QE} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\text{Real Arithmetic}}{\vdash \forall d \forall e, (2de - 2de = 0)}\,\text{QE}}{\vdash \forall d \forall e, (2de + 2e \cdot -d = 0)}}{\vdash \forall d \forall e, (2dd' + 2ee' = 0)^{e\ \ -d}_{d'\ e'}}}{\vdash \forall d \forall e, (d^2 + e^2 = r^2)'^{e\ \ -d}_{d'\ e'}}}{d = 0, e = 1, r = 1 \vdash [\dot{d} = e, \dot{e} = -d] d^2 + e^2 = r^2}\,[\text{DI}]$$

Figure 2.46: Proof tree of $d = 0, e = 1, r = 1 \vdash [\dot{d} = e, \dot{e} = -d] d^2 + e^2 = r^2$

2.4 Theoretical results

We state the soundness of the sequent calculus and its incompleteness. We present its relative completeness with respect to its discrete and continuous fragment. We present several sound extensions of $d\mathcal{L}$ and give a brief presentation of the theorem prover KeYmaera X.

Soundness theorem The *soundness theorem* means that if we have a derivation of a sequent, then the sequent is valid.

Theorem (Soundness theorem)**.** *If* $\Gamma \vdash \varphi \vee \Delta$*, then* $\Gamma \models \varphi \vee \Delta$.

Proof. We have already stated that the rules for the first-order real arithmetic formulas are sound. The interested reader can refer to [98, p.97] for a more detailed presentation of the soundness of other rules. □

This theorem is very important since it ensures that if we obtain a proof of a formula, then it is valid. The converse is not true. If a sequent is valid, there is not necessarily a derivation. As a consequence, the logic $d\mathcal{L}$ is incomplete.

Incompleteness theorem

Theorem (Incompleteness theorem)**.** *$d\mathcal{L}$ is incomplete, i.e. there exists a formula φ which is valid, but for whose it is impossible to derive a proof tree.*

Proof. We reduce this result to the halt problem. Since the discrete part of $d\mathcal{L}$ is Turing-complete, if we assume that it is complete, then the verification problem of programs is complete too. The halt problem is thus decidable which is absurd. □

Although the sequent calculus is not complete, we have relative completeness with respect to the discrete or continuous fragment.

Relative completeness Relative completeness of a proof calculus with respect of an other proof calculus means that if the first is complete, then the second too. The sequent calculus of $d\mathcal{L}$ is relatively complete with respect to both its discrete and continuous fragment [101].

Theorem (Relative completeness with respect to the discrete fragment)**.** *If we assume the discrete fragment to be complete, then the sequent calculus of $d\mathcal{L}$ is complete.*

Theorem (Relative completeness with respect to the continuous fragment)**.** *If we assume the continuous fragment to be complete, then the sequent calculus of $d\mathcal{L}$ is complete.*

These results show that reasoning on hybrid systems, purely discrete systems or purely continuous systems is equivalent from a proof-theoretical view.

Extensions Several extensions of $d\mathcal{L}$ have been proposed to address fundamental problems. Each of these extensions has a sound sequent calculus and has been implemented in KeYmaera.

Differential temporal logic (dTL) is a conservative extension with temporal constructs. It allows to reason about the temporal behavior during the execution of a hybrid system [96] [71]. It provides means to express more complex liveness specification.

Quantified differential dynamic logic ($\mathcal{Q}d\mathcal{L}$) is an extension to handle distributed hybrid systems [99]. *Differential dynamic game logic* ($dDG\mathcal{L}$) integrates several game constructs on top of $d\mathcal{L}$ allowing to analyze hybrid games [109]. Finally, *Stochastic differential dynamic logic* ($Sd\mathcal{L}$) is designed to handle stochastic hybrid systems [100]. Security and privacy concerns are addressed by introducing the notion of hybrid dynamic information flows. It allows to reason about leakage of informations both in cyber channels and physical channels.

KeYmaera and KeYmaera X $d\mathcal{L}$ has been first implemented in KeYmaera [105]. As $d\mathcal{L}$ is an extension of Dynamic logic, KeYmaera is an extension of the theorem prover Key which implements Dynamic logic [6].

It has been completely rewritten to form KeYmaera X [47] [85]. It features a small kernel which has been verified in Coq and Isabelle [25] and a graphical interface. There is also a proof programming language Bellerophon [46] which allows a user to define its own tactics.

Chapter 3

A modular component-based approach in Differential Dynamic Logic

The parallel composition of Cyber-Physical Systems (CPS) models the parallel execution of two CPS simultaneously. It is of paramount importance to scale to the modeling and proof of large systems since most of CPS run in parallel.

We present our methodology to modularly model and prove correctness of Cyber-Physical Systems. The Section 3.1 presents the definition of a component at the design level and its precise implementation into $d\mathcal{L}$. The Section 3.2 is devoted to the definition of a parallel composition operator which allows to modularly model a CPS. We show how to retain properties of a component through composition in Section 3.3. We have exemplified each notion with a cruise-controller example. The Section 3.4 present a prototype implemented in the theorem prover KeYmaera X. It is prototype to automate the proof procedure. The Section 3.5 details the complete study of a second use case, a water-tank.

3.1 Definition of a component

A *component* is a part of a system which has a distinct behavior and communicates with other parts of the system via inputs and outputs. Properties are expressed on inputs and outputs. It is a popular way to tackle the complexity of large systems, especially in the industry. The basic idea is to decompose a system into several sub-systems which are easier to understand and reason about.

The concept of component is fuzzy. It can denote the part of a mathematical proof, for example an auxiliary lemma that we prove independently, or a module in a software, for example the payment of a donation on Wikipedia. In the industry, a component may be the speed sensor or the power engine in a car.

We provide a clear representation for a component at the design level in Subsection 3.1.1 and its logical equivalence in $d\mathcal{L}$ in Subsection 3.1.2. We detail in Subsection 3.1.2 the general form that a component exhibits in $d\mathcal{L}$. We illustrate each notion and definition with a cruise-controller example inspired from the use-case in the approach of Mueller *et al.* [88].

3.1.1 What is a component

A component is made up of three parts: a *definition*, an *interface* and a *contract*. The definition provides a name and a description of the behavior of the component. The contract describes the requirements attached of the component. It gives information on the purpose of the component. The interface indicates how it can interact with the rest of the system. The three parts can be interpreted from a $d\mathcal{L}$ formula, *i.e.* it is possible to work purely from $d\mathcal{L}$ logic. However, we add a supplementary layer, called *textual representation*, to reason on the system without having to work on $d\mathcal{L}$ formulas which might get more and more complex along the construction of the system.

The conception of a system consists of two phases: a *design phase* and a *verifying phase*. The design phase requires good understanding of the domain in order to develop a meaningful system. The methodologies used here should not depend of a particular logic. The proving phase is more technical and may require deep knowledge of proof theory.

Environment We assume that the designer provides a description of the *environment* in the form of a set of formulas denoted by $\mathcal{E}$. It regroups the free variables that are not outputs of a component, the *parameters*. They cannot be controlled and are exterior to the considered

system. For example, the gravity value g is a parameter in most of mechanical systems or the value of the speed limit S in our example. It is a value fixed by the government, a tachymeter cannot change it freely. Parameters can be constant or in a range of value if we want to represent uncertainty.

Time We assume that the time evolves continuously and linearly. It is represented by the reserved variable t and its evolution is modeled by the differential equation $\dot{t} = 1$.

Name and behavior A component is defined by a *name* and a *behavior*. The behavior is the functional modeling of the component. It is represented as a hybrid program of $d\mathcal{L}$[1] and defines the evolution of the component trough time. We adopt the convention of denoting a generic component by a capital letter and the associated behavior with its equivalent in the greek alphabet.

Example 17 (Name and behavior of the `Tachymeter` component)**.** *The* `Tachymeter` *is a software which senses the speed of the vehicle every ε seconds. It decides the speed that should be attained by the vehicle after ε seconds and set up the acceleration given to the engine. Its behavior is defined by the hybrid program denoted* ***Tach****. We give a detailed presentation in Example 21.*

`Name: Tachymeter`
`Behavior:`

$$\textbf{Tach} \triangleq \Big(?t_{tach} + \varepsilon \geqslant t;\ s_{acm} := s_{eng};\ s_{tach} := *;\ ?(0 \leqslant s_{tach} \leqslant S \wedge -\delta \leqslant s_{tach} - s_{acm} \leqslant \delta);\ a := \frac{s_{tach} - s_{acm}}{\varepsilon};\ t_{tach} := t\Big)^*$$

Figure 3.1: Tachymeter

Interface The *interface* of a component describes how it interacts with the world. It specifies the *inputs* and the *outputs* along with the properties associated to every component.

Example 18 (Interface of the `Tachymeter`)**.** *The component* `Tachymeter` *possess one input and three outputs. The input s_{eng} represents the actual speed of the engine and the output s_{acm} the measured speed. s_{tach} is the target speed the vehicle should attain after ε seconds and a the acceleration to follow to reach the target speed.*

[1]The syntax of hybrid programs can be found in Definitions 1 and 3

```
Name: Tachymeter
Inputs:
```
s_{eng}

```
Outputs:
```
s_{acm}

s_{tach}

a

```
Behavior:
```
$$\textbf{Tach} \triangleq \big(?t_{tach} + \varepsilon \geqslant t;\ s_{acm} := s_{eng};\ s_{tach} := *;\ ?(0 \leqslant s_{tach} \leqslant S \wedge -\delta \leqslant s_{tach} - s_{acm} \leqslant \delta);\ a := \frac{s_{tach} - s_{acm}}{\varepsilon};\ t_{tach} := t\ \big)^*$$

Figure 3.2: Tachymeter with inputs and outputs

Contract We associate properties to a component by using *contracts*. A contract is a pair of formulas (A, G), called the *assumptions* A and the *guarantees* G. Guarantees are properties on the outputs of the component. Assumptions are properties that the inputs are presumed to satisfy.

We get inspiration from the meta-theory of contracts of Benveniste *et al.* [18]. The formulas are formulas of $d\mathcal{L}$[2].

Example 19 (Contract of the `Tachymeter`)**.** *We assume that the speed of the vehicle, s_{eng}, does not exceed the speed limit S. Under this assumption on the real speed, the tachymeter guarantees that the target speed does not exceed the speed limit. It also guarantees that the measured speed is inferior to S and links the value of the acceleration to the measured and the target speed.*

```
Name: Tachymeter
Inputs:
```
s_{eng}

```
Assumptions:
```
$0 \leqslant s_{eng} \leqslant S$

```
Outputs:
```
a

s_{tach}

s_{acm}

```
Guarantees:
```
$0 \leqslant s_{acm} \leqslant S \wedge 0 \leqslant s_{tach} \leqslant S \wedge a = \frac{s_{tach} - s_{acm}}{\varepsilon}$

```
Behavior:
```
$$\textbf{Tach} \triangleq \big(?t_{tach} + \varepsilon \geqslant t;\ s_{acm} := s_{eng};\ s_{tach} := *;\ ?(0 \leqslant s_{tach} \leqslant S \wedge -\delta \leqslant s_{tach} - s_{acm} \leqslant \delta);\ a := \frac{s_{tach} - s_{acm}}{\varepsilon};\ t_{tach} := t\ \big)^*$$

Figure 3.3: Complete representation of the tachymeter

We can also define the component `Engine` representing the speed of the vehicle regulated by the `Tachymeter` component.

[2] The syntax of formulas of $d\mathcal{L}$ can be found in Definition 8

Example 20 (`Engine` component)**.** *The inputs of the engine are the outputs of the tachymeter, and reciprocally the output s_{eng} is the input of the tachymeter. The assumptions of the engine are the guarantees of the tachymeter. It assumes that the targeted speed s_{tach} and the measured speed s_{acm} provided by the* `Tachymeter` *do not exceed the speed limit S. Reciprocally, the guarantees of* `Engine`*, i.e. that the actual speed s_{eng} does not exceed the speed limit, correspond to the assumptions of the* `Tachymeter` *component. The behavior **Engine** is the differential equation detailed later in Example 22.*

`Name: Engine`

`Inputs:`

a

s_{tach}

s_{acm}

`Assumptions:`

$0 \leqslant s_{acm} \leqslant S \wedge 0 \leqslant s_{tach} \leqslant S \wedge a = \frac{s_{tach} - s_{acm}}{\varepsilon}$

`Outputs:`

s_{eng}

`Guarantees:`

$0 \leqslant s_{eng} \leqslant S$

`Behavior:`

Engine $\triangleq \dot{s}_{eng} = a, \dot{t} = 1 \;\&\; 0 \leqslant t - t_{tach} \leqslant \varepsilon$

Figure 3.4: Textual definition of the Engine

The formulas A and G are usually denoted as pre- and post-conditions in the related work to Dynamic Logic [108] and Differential Dynamic Logic [98]. We have made the choice here to refer as assumptions and guarantees to emphasis that we develop a component-based approach.

Graphical representation Another popular way to represent a component is graphically. It is a compact representation and similar to existing design methods as dataflow models.

The interface is represented by incoming arrows for the inputs and outgoing arrows for the outputs. The box is divided in three part: the name `A`, the behavior α and the assumptions and guarantees (A, G). The graphical representation of the component `Engine` is provided in the Figure 3.5.

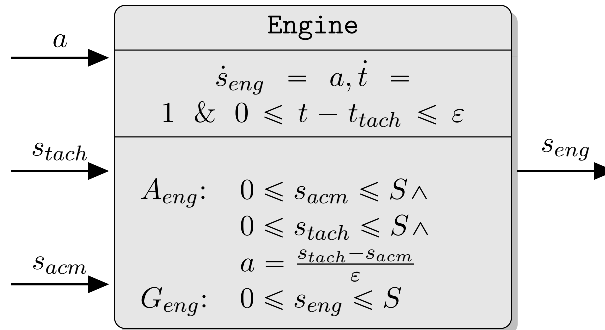


Figure 3.5: Graphical representation of the engine

Conclusion We have presented a concise representation of a component. It allow to reason on CPS without requiring a deep knowledge of $d\mathcal{L}$. It is very similar to the meta-theory of contract defined by Benveniste *et al.* [18]. We have instantiated with the logic $d\mathcal{L}$ in our approach, but one can imagine to mix it with other approaches.

3.1.2 Definition in $d\mathcal{L}$

We show how to transcript our notion of component into a formula of $d\mathcal{L}$. The behavior is already a hybrid program, but it should respect a particular form that we detail. The interface corresponds to bound and free variable of the behavior and the contract can be naturally expressed in $d\mathcal{L}$ as seen in Definition 10.

General form of a behavior in $d\mathcal{L}$ We define a general form that the behavior of a component should respect. To gain in clarity, we partition specifications α into their discrete and continuous parts $\mathbf{disc}_\alpha$ and $\mathbf{cont}_\alpha$.

Definition 19 (General form of a behavior)**.** *The general form of a behavior $\alpha \triangleq (\boldsymbol{cont}_\alpha \cup \boldsymbol{disc}_\alpha)^*$ consists of its partition into a continuous part $\boldsymbol{cont}_\alpha$, of the form $\dot{X} = \theta_X \ \& \ H_X$, and the discrete part $\boldsymbol{disc}_\alpha$.*

Notice that the discrete part of a system $\mathbf{disc}_\alpha$ is itself defined by the union of discrete, functional, components.

Example 21 (Behavior of **Tachymeter**)**.** *We model the discrete controller of a cruise control system that is responsible for delivering a targeted speed to the vehicle engine (e.g. a car).*

$$\begin{aligned}\boldsymbol{Tach} \ \triangleq \ & \big(?t_{tach} + \varepsilon \geqslant t; s_{acm} := s_{eng}; s_{tach} := *;\\ & ?(0 \leqslant s_{tach} \leqslant S \wedge -\delta \leqslant s_{tach} - s_{acm} \leqslant \delta);\\ & a := \tfrac{s_{tach} - s_{acm}}{\varepsilon}; t_{tach} := t\big)^*\end{aligned}$$

The test $?t_{tach} + \varepsilon \geqslant t$ ensure that no more that ε seconds have passed since the last execution marked by t_{tach}. We measure then the value of the speed s_{eng} by memorizing it in the variable s_{acm} ($s_{acm} := s_{eng}$. It chooses an arbitrary value for s_{ctrl}, the targeted speed, and checks if it is in the desired range $[0, S]$, where S is the speed limit. It additionally checks that the difference between the speed measured by the tachymeter, s_{tach}, and the targeted speed, s_{ctrl}, is not too high, in order to ensure that the acceleration set is not too brutal and/or within the capabilities of the engine.

Example 22 (Behavior of **Engine**)**.** *We model the continuous acceleration of the engine by a guarded derivative:*

$$\dot{s}_{eng} = a, \dot{t} = 1 \ \& \ 0 \leqslant t - t_{tach} \leqslant \varepsilon$$

The differential equation describes the evolution of speed by a function of the acceleration over time.

We can model a large variety of hybrid systems as sensors, programs, plant.

Contract of a component in $d\mathcal{L}$ To soundly ensure that a component satisfies its contract, we use the proof theory of $d\mathcal{L}$.

A component `A` in an environment $\mathcal{E}$ is translated in $d\mathcal{L}$ as a formula of the form $(\mathcal{E} \wedge A) \rightarrow [\alpha]G$ where $\mathcal{E}$ is the environment, A and G are the assumptions and guarantees and α the behavior of the component. For example, the component `Tachymeter` is translated as $(\mathcal{E} \wedge A_{tach}) \rightarrow [\textbf{Tach}]G_{tach}$.

The interface is implicit. The inputs (resp. outputs) of a component correspond to the *free variables* (resp. *bound variables*) of the behavior. We have `A.inputs` $\subseteq FV(\alpha)$ and `A.outputs` $\subseteq BV(\alpha)$. This notion is translated in $d\mathcal{L}$ by the notion of *bounds and free variables*. The bound variables of a hybrid program α are variables that can be changed during the execution of α, *i.e.* a variable on which the hybrid program acts. It is assimilated to outputs. The remaining variables are the free variables, *i.e.* all the variables that are not bound. The component cannot modify theses variables. We have to distinguish in these variables the *parameters*, which are part of the environment, and the inputs which are the variables that can potentially be outputs of other components.

It is now possible to precisely define the satisfaction of a contract by a component.

Definition 20 (Satisfaction of a contract)**.** *A component* `A` satisfies *its contract* (A, G) *if the formula* $(\mathcal{E} \wedge A) \rightarrow [\alpha]G$ *is valid.*

We add the environment E in the formula because it is needed to achieve the proof. We prove the validity of the formula by using rules of the sequent calculus of $d\mathcal{L}$ [98].

We refer to the behavior of the tachymeter by **Tach** in the next sections. We have:

$$\begin{aligned} \textbf{Tach} \ \triangleq\ & \big(?t_{tach} + \varepsilon \geqslant t; s_{acm} := s_{eng}; s_{tach} := *;\\ & ?(0 \leqslant s_{tach} \leqslant S \wedge -\delta \leqslant s_{tach} - s_{acm} \leqslant \delta);\\ & a := \tfrac{s_{tach} - s_{acm}}{\varepsilon}; t_{tach} := t\big)^* \end{aligned}$$

Example 23 (Satisfaction of the contract of the Tachymeter)**.** *To show that the* `Tachymeter` *component satisfies its contract, we prove the validity of the sequent:*

$$\mathcal{E}, A_{Tach} \vdash [\boldsymbol{Tach}]G_{Tach}$$

where:

$$\begin{aligned} \boldsymbol{Tach} &\triangleq (\boldsymbol{Body})^*\\ \boldsymbol{Body} &\triangleq ?t_{tach} + \varepsilon \geqslant t; s_{acm} := s_{eng}; s_{tach} := *;\\ &\quad ?(0 \leqslant s_{tach} \leqslant S \wedge -\delta \leqslant s_{tach} - s_{acm} \leqslant \delta);\\ &\quad a := \tfrac{s_{tach} - s_{acm}}{\varepsilon}; t_{tach} := t\\ A_{Tach} &\triangleq 0 \leqslant s_{eng} \leqslant S\\ G_{Tach} &\triangleq 0 \leqslant s_{acm} \leqslant S \wedge 0 \leqslant s_{tach} \leqslant S \wedge a = \tfrac{s_{tach} - s_{acm}}{\varepsilon}\\ \mathcal{E} &\triangleq S > 0 \wedge \varepsilon > 0 \wedge s_{tach} = s_{eng} \wedge s_{acm} = s_{eng} \wedge a = 0 \end{aligned}$$

We distinguish the hybrid program ***Body*** *in the loop from the behavior of the component* ***Tach*** *to clarify the proof.* ***Body*** *will occurs at multiple times. We use G_{Tach} as an invariant for the application of the rule [Ind].*

$$\dfrac{\dfrac{\Pi_{Init}}{\mathcal{E}, A_{Tach} \vdash G_{Tach}} \qquad \dfrac{\Pi_{Step}}{\mathcal{E}, A_{Tach} \vdash \forall s_{acm}, s_{tach}, a, t_{tach}(G_{Tach} \rightarrow [\boldsymbol{Body}]G_{Tach})}}{\mathcal{E}, A_{Tach} \vdash [(\boldsymbol{Body})^*]G_{Tach}}\ \textit{[Ind]}$$

We obtain two sub-goals left. The left goal is the initial step where we have to prove that G_{Tach} holds after zero execution of the Tachymeter. It is proved by the branch Π_{init}. The second is the induction step. We assume that G_{Tach} is true and we have to show that it still holds after one execution of the hybrid program ***Body****. We skolemize the formula by every bound variable of* ***Tach****. It is closed by the branch Π_{step} that we detail later.*

We deal first with the branch Π_{init}. We unfold the definition of G_{Tach} and decompose the goals in three sub-goals with the rule $\wedge_r$:

$$\dfrac{\dfrac{\mathcal{E}, A_{Tach} \vdash 0 \leqslant s_{acm} \leqslant S \qquad \dfrac{\mathcal{E}, A_{Tach} \vdash 0 \leqslant s_{tach} \leqslant S \qquad \mathcal{E}, A_{Tach} \vdash a = \frac{s_{tach}-s_{acm}}{\varepsilon}}{\mathcal{E}, A_{Tach} \vdash 0 \leqslant s_{tach} \leqslant S \wedge a = \frac{s_{tach}-s_{acm}}{\varepsilon}}\wedge_r}{\dfrac{\mathcal{E}, A_{Tach} \vdash 0 \leqslant s_{acm} \leqslant S \wedge 0 \leqslant s_{tach} \leqslant S \wedge a = \frac{s_{tach}-s_{acm}}{\varepsilon}}{\mathcal{E}, A_{Tach} \vdash G_{Tach}}\textit{Unfolding of } G_{Tach}}\wedge_r}{\Pi_{init}}$$

The three sub-goals $0 \leqslant s_{acm} \leqslant S$, $0 \leqslant s_{tach} \leqslant S$ and $a = \frac{s_{tach}-s_{acm}}{\varepsilon}$ are real arithmetic formulas which can be proved by an SMT solver. We detail the case of the first sub-goal; the two others are similar. It unfolds to the following sequent :

$$S > 0 \wedge \varepsilon > 0, s_{tach} = s_{eng}, s_{acm} = s_{eng}, a = 0, 0 \leqslant s_{eng} \leqslant S \vdash 0 \leqslant s_{acm} \leqslant S$$

We can replace s_{acm} by s_{eng} in the right-hand side of the sequent. We obtain $0 \leqslant s_{eng} \leqslant S$ in the right-hand side and $0 \leqslant s_{eng} \leqslant S$ is already present in the left-hand side. we conclude by applying the axiom rule.

We consider the induction step. We have to prove the sequent $\mathcal{E}, A_{Tach} \vdash \forall s_{acm}, s_{tach}, a, t_{tach}(G_{Tach} \rightarrow [\textbf{\textit{Body}}]G_{Tach})$. We apply the skolemization rule $\forall_r$ four times, denoted by $\forall_r^4$. We obtain $\mathcal{E}, A_{Tach} \vdash (G^0_{Tach} \rightarrow [\textbf{\textit{body}}^0]G^0_{Tach})$.

$$\dfrac{\mathcal{E}, A_{Tach} \vdash G^0_{Tach} \rightarrow [\textbf{\textit{Body}}^0]G^0_{Tach}}{\mathcal{E}, A_{Tach} \vdash \forall s_{acm}, s_{tach}, a, t_{tach}(G_{Tach} \rightarrow [\textbf{\textit{Body}}]G_{Tach})}\forall_r^4$$

G^0_{Tach} (resp. ***Body***^{0}*) is G_{Tach} where every occurrences of $s_{acm}, s_{tach}, a, t_{tach}$ have been replaced by the fresh variables $s^0_{acm}, s^0_{tach}, a^0, t^0_{tach}$ (resp.* ***Body***^{0}*). We have:*

$$\begin{array}{rcl} \textbf{\textit{Body}}^0 & \triangleq & ?t^0_{tach} + \varepsilon \geqslant t; s^0_{acm} := s_{eng}; s^0_{tach} := *; \\ & & ?(0 \leqslant s^0_{tach} \leqslant S \wedge -\delta \leqslant s^0_{tach} - s^0_{acm} \leqslant \delta); \\ & & a^0 := \frac{s^0_{tach}-s^0_{acm}}{\varepsilon}; t^0_{tach} := t \\ G^0_{Tach} & \triangleq & 0 \leqslant s^0_{acm} \leqslant S \wedge 0 \leqslant s^0_{tach} \leqslant S \wedge a^0 = \frac{s^0_{tach}-s^0_{acm}}{\varepsilon} \end{array}$$

We use the rule [BoxAnd] twice to split the conjunctions in the invariant.

$$\underbrace{\dfrac{\mathcal{E}, A_{Tach}, G^0_{Tach} \vdash [\textbf{\textit{Body}}^0]0 \leqslant s^0_{tach} \leqslant S \qquad \mathcal{E}, A_{Tach}, G^0_{Tach} \vdash [\textbf{\textit{Body}}^0]a^0 = \frac{s^0_{tach}-s^0_{acm}}{\varepsilon}}{\mathcal{E}, A_{Tach}, G^0_{Tach} \vdash [\textbf{\textit{Body}}^0]0 \leqslant s^0_{tach} \leqslant S \wedge a^0 = \frac{s^0_{tach}-s^0_{acm}}{\varepsilon}}\textit{[BoxAnd]}}$$

$$\dfrac{\dfrac{\dfrac{\mathcal{E}, A_{Tach}, G^0_{Tach} \vdash [\textbf{\textit{Body}}^0]0 \leqslant s^0_{acm} \leqslant S \qquad \uparrow}{\mathcal{E}, A_{Tach}, G^0_{Tach} \vdash [\textbf{\textit{Body}}^0](0 \leqslant s^0_{acm} \leqslant S \wedge 0 \leqslant s^0_{tach} \leqslant S \wedge a^0 = \frac{s^0_{tach}-s^0_{acm}}{\varepsilon})}\textit{[BoxAnd]}}{\mathcal{E}, A_{Tach} \vdash G^0_{Tach} \rightarrow [\textbf{\textit{Body}}^0]G^0_{Tach}}\textit{Unfold } G^0_{Tach}}{\mathcal{E}, A_{Tach} \vdash G^0_{Tach} \rightarrow [\textbf{\textit{Body}}^0]G^0_{Tach}}\rightarrow_r$$

*We detail the proof of the sub-goal $\mathcal{E}, A_{Tach}, G^0_{Tach} \vdash [\boldsymbol{Body}^0] 0 \leqslant s^0_{acm} \leqslant S$ in Figure 3.6. The two other sub-goals are similar. The basic idea is to unfold **Body**, the resulting sequent is composed of Real Arithmetic formulas and is trivial to prove.*

The corresponding .kyx file can be found in the annexes 5.3.2 for the `Tachymeter` and 5.3.2 for the `Engine`.

Conclusion We have presented how to represent a component `A` with its interface, contract and behavior. We have defined the meaning of `A` satisfying its contract. We have explained how it is translated in the $d\mathcal{L}$ framework. It gives us a strong logical basis to soundly reason on CPS. The next section is devoted to the presentation of the process of parallel composition to modularly build a system.

3.2 Parallel composition operator

The composition of components is the core aspect of the component-based approach; it is the mechanism that allow us to obtain a full system after having considered each of its components separately. It specifies how we connect components and what the resulting behavior is.

The operator achieving it shall be the more general possible in order to handle a wide variety of systems. The result of the composition of two components, the *composite*, shall be a component that we can composed later with another component. The operator must be modular, *i.e.* commutative and associative. If we compose two components `A` and `B`, then another one `C`, we shall obtain the same result as if we have composed first `B` and `C` together, and then `A` as illustrated in the Figure 3.7.

We present the parallel composition of components `A` and `B` in Subsection 3.2.1. The Subsection 3.2.2 is devoted to the definition of the operator $\circ$ to compose behaviors in $d\mathcal{L}$. We discuss the differences with the parallel composition of hybrid action systems in Subsection 3.2.3.

3.2.1 Parallel composition of components

When composing two components, there is no need to specify the inputs, the assumptions, the outputs, the guarantees and the behavior of the resulting component; these characteristics are automatically inherited from the composition. The inputs (resp. outputs) are the union of respective inputs (resp. outputs). The assumptions (resp. guarantees) are the conjunction of respective assumptions (resp. guarantees). The behavior is obtained by application of the operator $\circ$, defined in Subsection 3.2.2.

Our model of communication for our parallel composition operator is based on variable-sharing and is thus very simple. An output can be the input of several components, and it is thus not possible to abstract matching inputs and outputs through composition. We outline how we can model communication channels in our paradigm in the Subsection 5.1.3 of future works.

Example 24 (Cruise-Control)**.** *The cruise-control system is obtained by parallel composition of components* `Tachymeter` *and* `Engine`*. It is itself a component, named* `Cruise-Control` *that may be composed later with other part of a vehicle, e.g. a power-train. The result of the composition is given in Figure 3.8.*

A graphical representation is given in the Figure 3.9.

$$\dfrac{}{\mathcal{E}, 0 \leqslant s^0_{eng} \leqslant S, G^0_{Tach}, t^0_{tach} + \varepsilon \geqslant t, 0 \leqslant x \leqslant S \wedge -\delta \leqslant x - s^0_{eng} \leqslant \delta \vdash 0 \leqslant s^0_{eng} \leqslant S} \text{ ax}$$

Unfold definition of A_{Tach}

$$\mathcal{E}, A_{Tach}, G^0_{Tach}, t^0_{tach} + \varepsilon \geqslant t, 0 \leqslant x \leqslant S \wedge -\delta \leqslant x - s^0_{eng} \leqslant \delta \vdash 0 \leqslant s^0_{eng} \leqslant S$$

[:=]

$$\mathcal{E}, A_{Tach}, G^0_{Tach}, t^0_{tach} + \varepsilon \geqslant t, 0 \leqslant x \leqslant S \wedge -\delta \leqslant x - s^0_{eng} \leqslant \delta \vdash [t^0_{tach} := t]0 \leqslant s^0_{eng} \leqslant S$$

[:=]

$$\mathcal{E}, A_{Tach}, G^0_{Tach}, t^0_{tach} + \varepsilon \geqslant t, 0 \leqslant x \leqslant S \wedge -\delta \leqslant x - s^0_{eng} \leqslant \delta \vdash [a^0 := \tfrac{x - s^0_{eng}}{\varepsilon}][t^0_{tach} := t]0 \leqslant s^0_{eng} \leqslant S$$

[;]

$$\mathcal{E}, A_{Tach}, G^0_{Tach}, t^0_{tach} + \varepsilon \geqslant t, 0 \leqslant x \leqslant S \wedge -\delta \leqslant x - s^0_{eng} \leqslant \delta \vdash [a^0 := \tfrac{x - s^0_{eng}}{\varepsilon}; t^0_{tach} := t]0 \leqslant s^0_{eng} \leqslant S$$

$[\rightarrow_r]$

$$\mathcal{E}, A_{Tach}, G^0_{Tach}, t^0_{tach} + \varepsilon \geqslant t \vdash (0 \leqslant x \leqslant S \wedge -\delta \leqslant x - s^0_{eng} \leqslant \delta) \rightarrow [a^0 := \tfrac{x - s^0_{eng}}{\varepsilon}; t^0_{tach} := t]0 \leqslant s^0_{eng} \leqslant S$$

[?]

$$\mathcal{E}, A_{Tach}, G^0_{Tach}, t^0_{tach} + \varepsilon \geqslant t \vdash [?(0 \leqslant x \leqslant S \wedge -\delta \leqslant x - s^0_{eng} \leqslant \delta)][a^0 := \tfrac{x - s^0_{eng}}{\varepsilon}; t^0_{tach} := t]0 \leqslant s^0_{eng} \leqslant S$$

[;]

$$\mathcal{E}, A_{Tach}, G^0_{Tach}, t^0_{tach} + \varepsilon \geqslant t \vdash [?(0 \leqslant x \leqslant S \wedge -\delta \leqslant x - s^0_{eng} \leqslant \delta); a^0 := \tfrac{x - s^0_{eng}}{\varepsilon}; t^0_{tach} := t]0 \leqslant s^0_{eng} \leqslant S$$

$[\forall_r]$

$$\mathcal{E}, A_{Tach}, G^0_{Tach}, t^0_{tach} + \varepsilon \geqslant t \vdash \forall x([?(0 \leqslant x \leqslant S \wedge -\delta \leqslant x - s^0_{eng} \leqslant \delta); a^0 := \tfrac{x - s^0_{eng}}{\varepsilon}; t^0_{tach} := t]0 \leqslant s^0_{eng} \leqslant S)$$

[;]

$$\mathcal{E}, A_{Tach}, G^0_{Tach}, t^0_{tach} + \varepsilon \geqslant t \vdash [s^0_{tach} := *][?(0 \leqslant s^0_{tach} \leqslant S \wedge -\delta \leqslant s^0_{tach} - s^0_{eng} \leqslant \delta); a^0 := \tfrac{s^0_{tach} - s^0_{eng}}{\varepsilon}; t^0_{tach} := t]0 \leqslant s^0_{eng} \leqslant S$$

[;]

$$\mathcal{E}, A_{Tach}, G^0_{Tach}, t^0_{tach} + \varepsilon \geqslant t \vdash [s^0_{tach} := *; ?(0 \leqslant s^0_{tach} \leqslant S \wedge -\delta \leqslant s^0_{tach} - s^0_{eng} \leqslant \delta); a^0 := \tfrac{s^0_{tach} - s^0_{eng}}{\varepsilon}; t^0_{tach} := t]0 \leqslant s^0_{eng} \leqslant S$$

[:=]

$$\mathcal{E}, A_{Tach}, G^0_{Tach}, t^0_{tach} + \varepsilon \geqslant t \vdash [s^0_{acm} := s_{eng}][s^0_{tach} := *; ?(0 \leqslant s^0_{tach} \leqslant S \wedge -\delta \leqslant s^0_{tach} - s^0_{acm} \leqslant \delta); a^0 := \tfrac{s^0_{tach} - s^0_{acm}}{\varepsilon}; t^0_{tach} := t]0 \leqslant s^0_{acm} \leqslant S$$

[;]

$$\mathcal{E}, A_{Tach}, G^0_{Tach}, t^0_{tach} + \varepsilon \geqslant t \vdash [s^0_{acm} := s_{eng}; s^0_{tach} := *; ?(0 \leqslant s^0_{tach} \leqslant S \wedge -\delta \leqslant s^0_{tach} - s^0_{acm} \leqslant \delta); a^0 := \tfrac{s^0_{tach} - s^0_{acm}}{\varepsilon}; t^0_{tach} := t]0 \leqslant s^0_{acm} \leqslant S$$

$\rightarrow_r$

$$\mathcal{E}, A_{Tach}, G^0_{Tach} \vdash t^0_{tach} + \varepsilon \geqslant t \rightarrow [s^0_{acm} := s_{eng}; s^0_{tach} := *; ?(0 \leqslant s^0_{tach} \leqslant S \wedge -\delta \leqslant s^0_{tach} - s^0_{acm} \leqslant \delta); a^0 := \tfrac{s^0_{tach} - s^0_{acm}}{\varepsilon}; t^0_{tach} := t]0 \leqslant s^0_{acm} \leqslant S$$

[?]

$$\mathcal{E}, A_{Tach}, G^0_{Tach} \vdash [?t^0_{tach} + \varepsilon \geqslant t][s^0_{acm} := s_{eng}; s^0_{tach} := *; ?(0 \leqslant s^0_{tach} \leqslant S \wedge -\delta \leqslant s^0_{tach} - s^0_{acm} \leqslant \delta); a^0 := \tfrac{s^0_{tach} - s^0_{acm}}{\varepsilon}; t^0_{tach} := t]0 \leqslant s^0_{acm} \leqslant S$$

[;]

$$\mathcal{E}, A_{Tach}, G^0_{Tach} \vdash [?t^0_{tach} + \varepsilon \geqslant t; s^0_{acm} := s_{eng}; s^0_{tach} := *; ?(0 \leqslant s^0_{tach} \leqslant S \wedge -\delta \leqslant s^0_{tach} - s^0_{acm} \leqslant \delta); a^0 := \tfrac{s^0_{tach} - s^0_{acm}}{\varepsilon}; t^0_{tach} := t]0 \leqslant s^0_{acm} \leqslant S$$

$$\mathcal{E}, A_{Tach}, G^0_{Tach} \vdash [\mathbf{Body}^0]0 \leqslant s^0_{acm} \leqslant S$$

Figure 3.6: Proof that the measured speed s_{acm} stays in the pre-defined range after one execution of the `Tachymeter`

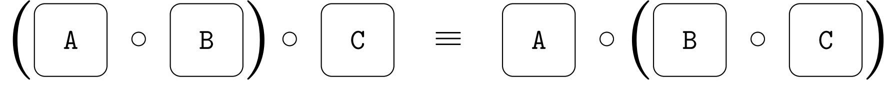


Figure 3.7: Modularity of composition

`Component: Cruise-Control`
`Inputs:`
s_{acm}
s_{tach}
a
s_{eng}
`Assumptions:`
$A_{tach} \wedge A_{eng}$
`Outputs:`
s_{acm}
s_{tach}
a
s_{eng}
`Guarantees:`
$G_{tach} \wedge G_{eng}$
`Behavior:`
Engine $\circ$ **Tach**

Figure 3.8: Cruise-control defined by composition of `Engine` and `Tachymeter`

3.2.2 Definition in $d\mathcal{L}$

We present in first part the *parallel continuous composition operator* to compose purely continuous behaviors of component, *i.e.* Ordinary Differential Equations (ODEs). We consider then the parallel composition operator for behaviors under the general form as in Definition 19.

Our parallel composition is largely inspired by the work of Ronkko *et al.* [114]. We compare the two approaches in the next Subsection 3.2.3.

A continuous parallel composition operator

We define the parallel composition $\circ_c$ between two continuous behaviors of a component. Recall that a continuous behavior α is a simple ODE, *i.e.* of the form $\dot{X} = \Theta_X \ \& \ H_X$.

Definition 21 (Definition of $\circ_c$). *Let* $\alpha \triangleq (\dot{X} = \theta_X \ \& \ H_X)$ *and* $\beta \triangleq (\dot{Y} = \theta_Y \ \& \ H_Y)$ *be continuous behaviors of components* `A` *and* `B`. *Assume that* X *and* Y *are separated, i.e.* $X \cap Y = \varnothing$. *The parallel composition is:*

$$\alpha \circ_c \beta \triangleq (\dot{X} = \Theta_X, \dot{Y} = \Theta_Y \ \& \ H_X \wedge H_Y)$$

It is the system composed of the two differential equations in parallel. The resulting evolution domain is the conjunction of respective evolution domains. We require that the

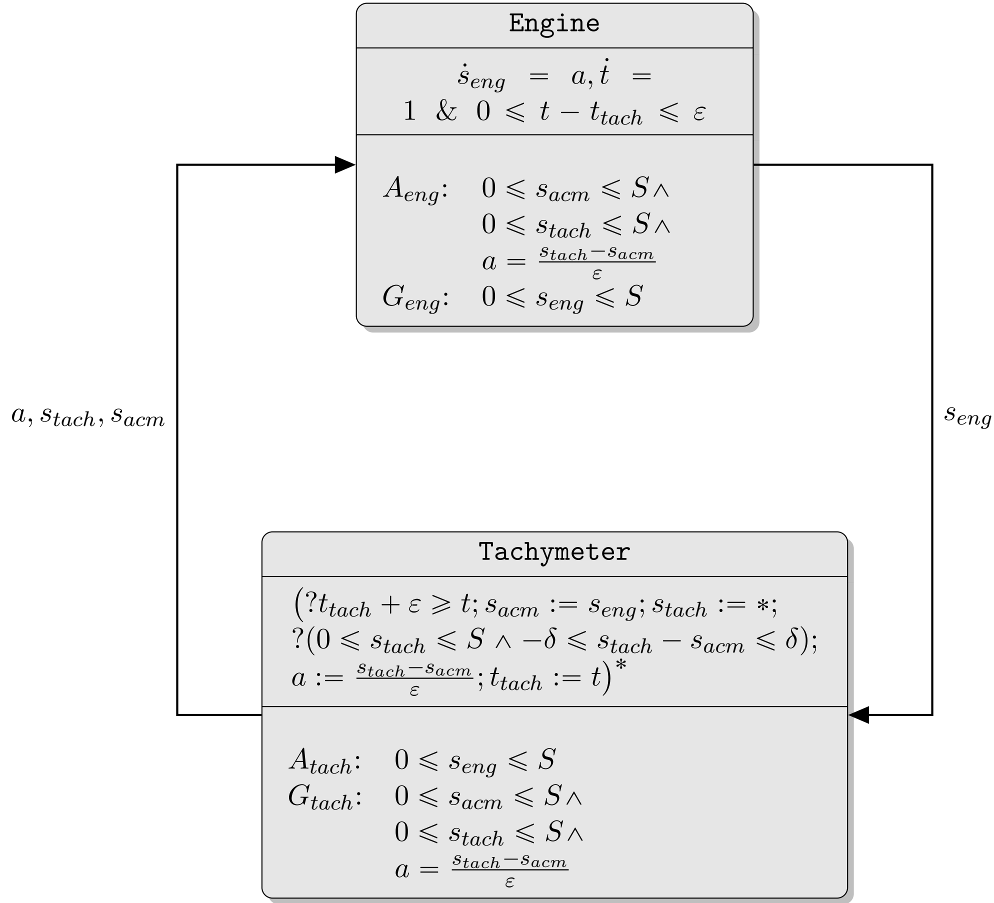


Figure 3.9: Cruise-control system

derived variables are separated ($X \cap Y = \emptyset$), *i.e.* that they do not describe the evolution of the same system. It is a restriction that makes sense at the design level, but it is also mandatory to lift the contracts through composition. But occurrences of variables in X (resp. in Y) can occur in Θ_Y (resp. in Θ_X).

To illustrate the operator, assume that we want to set a time limit to our vehicle engine component.

Example 25 (Time limit to the engine vehicle)**.** *Assume that in the description of the engine, we did not make explicit the time and we do not have set a limit. We do not want to consider perpetual execution of* `Engine` *since the tachymeter controls it periodically. To obtain the behavior of the component* `Engine` *in 22, we consider the time as a component with output the variable t, and the representation of the passing of time is achieved by introducing the ODE $\dot{t} = 1$. We add the formula $0 \leqslant t - t_{tach} \leqslant \varepsilon$ in the evolution domain of $\dot{t} = 1$, where ε is the desired limit. Here, we limit the time with the formula $0 \leqslant t - t_{tach} \leqslant \varepsilon$ rather than $t \leqslant \varepsilon$. It limits the time passed since the last execution of the* `Tachymeter` *component.*

By composing this proposition with our example, we obtain the timed model:

$$(\dot{s}_{eng} = a) \circ_c (\dot{t} = 1 \ \ \& \ \ 0 \leqslant t - t_{tach} \leqslant \varepsilon) \triangleq (\dot{s_{eng}} = a, \dot{t} = 1 \ \ \& \ \ 0 \leqslant t - t_{tach} \leqslant \varepsilon)$$

We may also compose the speed with other physical parts of the vehicle like the temper-

ature of the engine or the fuel's consumption.

Algebraic properties The parallel continuous composition operator $\circ_c$ is commutative and associative. Commutativity means that we can compose two physical systems in any order. Associativity means that we can build a system step-by-step; we can compose two components together, add a third component later, and we still obtain the same system as if we have compose all together in the first place. It allows to hierarchy the composition of components.

Proposition 1 (Commutativity of $\circ_c$). *Let α and β be continuous behaviors of components* `A` *and* `B`*.*

$$\alpha \circ_c \beta = \beta \circ_c \alpha$$

Proof. The idea is that the order of variables in an ODE is not important, we can re-arrange them to show associativity.

Let α and β be continuous behaviors of components `A` and `B`, *i.e.* $\alpha \triangleq \dot{X} = \theta_X \;\&\; H_X$ and $\beta \triangleq \dot{Y} = \theta_Y \;\&\; H_Y$.

$$\begin{array}{lr}
(\dot{X} = \theta_X \;\&\; H_X) \circ_c (\dot{Y} = \theta_Y \;\&\; H_Y) & \\
\triangleq \dot{X}, \dot{Y} = \theta_X, \theta_Y \;\&\; (H_X \wedge H_Y) & \text{(definition of } \circ_c \text{)} \\
\triangleq \dot{Y}, \dot{X} = \theta_Y, \theta_X \;\&\; (H_X \wedge H_Y) & \text{(re-arrangement)} \\
\triangleq \dot{Y}, \dot{X} = \theta_Y, \theta_X \;\&\; (H_Y \wedge H_X) & \text{(commutativity of } \wedge \text{)} \\
\triangleq (\dot{Y} = \theta_Y \;\&\; H_Y) \circ_c (\dot{X} = \theta_X \;\&\; H_X) & \text{(definition of } \circ_c \text{)}
\end{array}$$

□

Proposition 2 (Associativity of $\circ_c$). *Let α, β and γ be continuous behaviors of components* `A`*,* `B` *and* `C`*.*

$$(\alpha \circ_c \beta) \circ_c \gamma = \alpha \circ_c (\beta \circ_c \gamma)$$

Proof. As for the proof of commutativity, the order of variables in an ODE is not important; we can re-arrange them to show associativity.

Let α, β and γ be continuous behaviors of components `A`, `B` and `C`, *i.e.* $\alpha \triangleq \dot{X} = \theta_X \;\&\; H_X$, $\beta \triangleq \dot{Y} = \theta_Y \;\&\; H_Y$ and $\gamma \triangleq \dot{Z} = \theta_Z \;\&\; H_Z$.

$$\begin{array}{lr}
(\alpha \circ_c \beta) \circ_c \gamma & \\
\triangleq (\dot{X}, \dot{Y} = \theta_X, \theta_Y \;\&\; H_X \wedge H_Y) \circ_c \gamma & \text{(unfold definition of } \circ_c) \\
\triangleq \dot{X}, \dot{Y}, \dot{Z} = \theta_X, \theta_Y, \theta_Z \;\&\; H_X \wedge H_Y \wedge H_Z & \text{(unfold definition of } \circ_c) \\
\triangleq \alpha \circ_c (\dot{Y}, \dot{Z} = \theta_Y, \theta_Z \;\&\; H_Y \wedge H_Z) & \text{(fold definition of } \circ_c) \\
\triangleq \alpha \circ_c (\beta \circ_c \gamma) & \text{(fold definition of } \circ_c)
\end{array}$$

□

We have presented the parallel composition operator for continuous behaviors of components. We have shown that it is associative and commutative, two mandatory conditions for the modularity. We present in the next section the parallel composition operator for behaviors of the general form.

A parallel composition operator

Recall that a behavior α is said of the general form if $\alpha \triangleq (\mathbf{disc}_\alpha \cup \mathbf{cont}_\alpha)^*$ (cf Definition 19. We define the parallel composition between such behaviors. The behavior is divided between its discrete part $\mathbf{disc}_\alpha$ and continuous part $\mathbf{cont}_\alpha$. Due to their different nature, the composition between discrete parts of components differs from the composition between their continuous part.

Definition 22 (Parallel composition between general behaviors)**.** *Let α and β be general behaviors of components* `A` *and* `B`*, i.e. $\alpha \triangleq (\boldsymbol{disc}_\alpha \cup \boldsymbol{cont}_\alpha)^*$ and $\beta \triangleq (\boldsymbol{disc}_\beta \cup \boldsymbol{cont}_\beta)^*$. The parallel composition operator $\circ$ is defined by:*

$$\alpha \circ \beta \triangleq \big(\boldsymbol{disc}_\alpha \cup \boldsymbol{disc}_\beta \cup (\boldsymbol{cont}_\alpha \circ_c \boldsymbol{cont}_\beta)\big)^*$$

where $\circ_c$ is defined in Definition 21.

The composition between the discrete parts is a non-deterministic choice, which amounts to an interleaving of every discrete parts. This full interleaving allows to model parallel composition of discrete parts of hybrid systems. The continuous part are composed with $\circ_c$. the discrete and continuous dynamics are then composed with the non-deterministic choice $\cup$. Two examples of executions are given in the Figure 3.10.

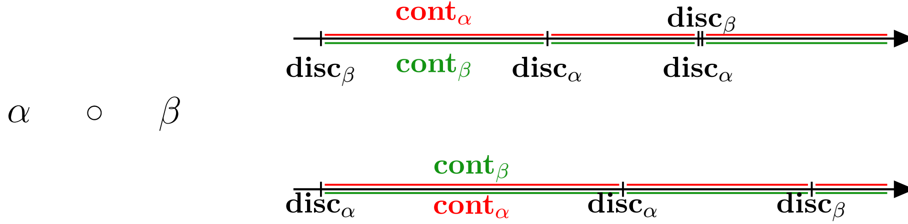


Figure 3.10: Possible executions of the parallel composition

We exemplify the parallel composition of behaviors with both discrete and continuous part by considering the composition of the `Engine`'s behavior with the `Tachymeter`'s behavior.

Example 26 (Behavior of the `Cruise-control`)**.** *The* `Cruise-control` *component is made up of the* `Tachymeter` *and the* `Engine`*. It is itself a component that can be composed with other components like the fuel consumption monitor. Its behavior* ***Cruise-control*** *is obtained by the composition of* ***Engine*** *and* ***Tach****.*

$$\begin{aligned}&\boldsymbol{Cruise\text{-}control} \triangleq \boldsymbol{Engine} \circ \boldsymbol{Tach} \triangleq\\ &\Big(\big(?t_{tach} + \varepsilon \geqslant t; s_{acm} := s_{eng}; s_{tach} := *; ?(0 \leqslant s_{tach} \leqslant S \wedge -\delta \leqslant s_{tach} - s_{acm} \leqslant \delta);\\ &a := \tfrac{s_{tach} - s_{acm}}{\varepsilon}; t_{tach} := t\big)\\ &\cup \big(\dot{s_{eng}} = a, \dot{t} = 1 \ \ \& \ \ 0 \leqslant t - t_{tach} \leqslant \varepsilon\big)\Big)^*\end{aligned}$$

The tachymeter's behavior senses the speed s_{eng} and decides of the targeted speed s_{tach}. It set the acceleration a followed by the engine's behavior. The test $t_{tach} + \varepsilon \geqslant t$ at the beginning of ***Tach*** *ensures that there is at most ε seconds between two execution of the tachymeter. t_{tach} records the last instant the controller have executed.*

Algebraic properties As in the continuous case, the parallel composition operator $\circ$ is commutative and associative.

Proposition 3 (Commutativity)**.** *Let α and β be behaviors of components* `A` *and* `B`*.*

$$\alpha \circ \beta = \beta \circ \alpha$$

Proof. The commutativity property results from the commutativity of the operator $\cup$ and the commutativity of the operator $\circ_c$. Let α and β be behaviors of components `A` and `B`, *i.e.* $\alpha \triangleq (\mathbf{disc}_\alpha \cup \mathbf{cont}_\alpha)^*$ and $\beta \triangleq (\mathbf{disc}_\beta \cup \mathbf{cont}_\beta)^*$. We have:

$$\begin{array}{rcll}
\alpha \circ \beta & \triangleq & \big(\mathbf{disc}_\alpha \cup \mathbf{disc}_\beta \cup (\mathbf{cont}_\alpha \circ_c \mathbf{cont}_\beta)\big)^* & \text{(Unfold definition of } \circ) \\
& \triangleq & \big(\mathbf{disc}_\beta \cup \mathbf{disc}_\alpha \cup (\mathbf{cont}_\alpha \circ_c \mathbf{cont}_\beta)\big)^* & \text{(Commutativity of } \cup) \\
& \triangleq & \big(\mathbf{disc}_\beta \cup \mathbf{disc}_\alpha \cup (\mathbf{cont}_\beta \circ_c \mathbf{cont}_\alpha)\big)^* & \text{(Commutativity of } \circ_c) \\
& \triangleq & \beta \circ \alpha & \text{(Fold definition of } \circ)
\end{array}$$

□

Proposition 4 (Associativity)**.** *Let α, β and γ be behaviors of components* `A`*,* `B` *and* `C`*.*

$$(\alpha \circ \beta) \circ \gamma = \alpha \circ (\beta \circ \gamma)$$

Proof. The associativity property results from the associativity of the operator $\cup$ and the associativity of the operator $\circ_c$.

Let α, β and γ be behaviors of components `A`, `B` and `C`, *i.e.* $\alpha \triangleq (\mathbf{disc}_\alpha \cup \mathbf{cont}_\alpha)^*$, $\beta \triangleq (\mathbf{disc}_\beta \cup \mathbf{cont}_\beta)^*$ and $\gamma \triangleq (\mathbf{disc}_\gamma \cup \mathbf{cont}_\gamma)^*$.

$$\begin{array}{cll}
& (\alpha \circ \beta) \circ \gamma & \\
\triangleq & \Big(\big(\mathbf{disc}_\alpha \cup \mathbf{disc}_\beta \cup (\mathbf{cont}_\alpha \circ_c \mathbf{cont}_\beta)\big)^*\Big) \circ \gamma & \text{(Unfold definition of } \circ) \\
\triangleq & \Big((\mathbf{disc}_\alpha \cup \mathbf{disc}_\beta) \cup \mathbf{disc}_\gamma \cup \big((\mathbf{cont}_\alpha \circ_c \mathbf{cont}_\beta) \circ_c \mathbf{cont}_\gamma\big)\Big)^* & \text{(Unfold definition of } \circ) \\
\triangleq & \Big(\mathbf{disc}_\alpha \cup (\mathbf{disc}_\beta \cup \mathbf{disc}_\gamma) \cup \big((\mathbf{cont}_\alpha \circ_c \mathbf{cont}_\beta) \circ_c \mathbf{cont}_\gamma\big)\Big)^* & \text{(Associativity of } \cup) \\
\triangleq & \Big(\mathbf{disc}_\alpha \cup (\mathbf{disc}_\beta \cup \mathbf{disc}_\gamma) \cup \big(\mathbf{cont}_\alpha \circ_c (\mathbf{cont}_\beta \circ_c \mathbf{cont}_\gamma)\big)\Big)^* & \text{(Associativity of } \circ_c) \\
\triangleq & \alpha \circ \Big(\big(\mathbf{disc}_\beta \cup \mathbf{disc}_\gamma \cup (\mathbf{cont}_\beta \circ_c \mathbf{cont}_\gamma)\big)^*\Big) & \text{(Fold definition of } \circ) \\
\triangleq & \alpha \circ (\beta \circ \gamma) & \text{(Fold definition of } \circ)
\end{array}$$

□

Conclusion We have defined a parallel composition operator for Cyber-Physical Systems. It is syntactically defined, and thus amenable to automation. It enjoys two important algebraic properties: commutativity and associativity. The associativity property allows to consider the construction of a system step-by-step.

3.2.3 Comparison with the parallel composition of hybrid action systems

We compare our parallel composition operator $\circ$ with the parallel composition for hybrid systems presented by Ronkko *et al.* [114]. We briefly present the hybrid action systems. The second part is devoted to a detailed comparison of both approaches.

Hybrid action systems We first define what is a *hybrid action system*. It is used to model reactive components, possibly non-terminating.

Definition 23 (Action system)**.** *An* action system *is an initialized block of the form:*

$$\begin{array}{lll} \mathcal{A} = & |[& \textbf{\textit{var}}\ X : T \\ & & X := E; \\ & & \textbf{\textit{do}}\ A_1 \| \ldots \| A_n\ \textbf{\textit{od}} \\ &]| & : Z \end{array}$$

var $X : T$ declares the list of variables X of type T. The variables can be global or local. Global variables can be shared with other action systems. Z declares the imported global variables that are used in $A_1, \ldots, A_n$, but do not belong to X. The action $X := E$ initializes the variables X by expressions E. After the initialization, the actions $A_1, \ldots, A_n$ are repeatedly executed when it is possible. There is no fairness assumption and the choice is non-deterministic.

Actions can be assignment $X := E$, sequence $A; B$, non-deterministic choice $\|$, or a guard $p \to A$ where p is a predicate and A an action.

Example 27 (Tachymeter)**.** *The tachymeter example presented in the Example 21 is represented by the following action system:*

$$\begin{array}{lll} \mathcal{TACH} = & |[& \textbf{\textit{var}}\ t_{tach} : \mathbb{R}, s_{acm} : \mathbb{R}, s_{tach} : \mathbb{R}, a : \mathbb{R} \\ & & s_{acm} := s_{eng}, s_{tach} := s_{eng}, a := 0; \\ & & \textbf{\textit{do}}\ t_{tach} + \varepsilon \geqslant t \to \big(s_{acm} := s_{eng}; s_{tach} := *; \\ & & (0 \leqslant s_{tach} \leqslant S \wedge -\delta \leqslant s_{tach} - s_{acm} \leqslant \delta) \to \\ & & (a := \frac{s_{tach} - s_{acm}}{\varepsilon}; t_{tach} := t)\big)\ \textbf{\textit{od}} \\ &]| & : s_{eng}, \varepsilon, t, S, \delta \end{array}$$

We first declare the variables $t_{tach}, s_{acm}, s_{tach}, a$ since they are the bounds variables of the system and type them with $\mathbb{R}$. Then we initialize them. The specification is very similar to our definition in Example 21. The two differences are that the test $?t_{tach} + \varepsilon \geqslant t; \ldots$ is written $t_{tach} + \varepsilon \geqslant t \to \ldots$ and the iteration symbol * is not explicit since the semantic of ***do**... **od*** *is the one of a while loop. Finally, we have to declare the variables $s_{eng}, \varepsilon, t, S, \delta$ as imported.*

Definition 24 (Differential action)**.** *A* differential action *is of the form $e : \dot{X} = F(X)$ where e is a guard and $\dot{X} = F(X)$ is a system of differential equations.*

The notation in $d\mathcal{L}$ would be $\dot{X} = F(X)\ \ \&\ \ e$.

Definition 25 (Hybrid action system)**.** *A* hybrid action system *is an initialized block of the form:*

$$\begin{array}{lll} \mathcal{H} = & |[& \textbf{\textit{var}}\ X : T \\ & & X := I; \\ & & \textbf{\textit{alt}}\ D\ \textbf{\textit{with}}\ DA \\ &]| & : Z \end{array}$$

alt D **with** DA is equivalent to **do** $gD \to D \| \neg gD \to DA$ **od**. All the discrete actions D must execute before the differential action may execute. Thus, discrete changes have a priority over continuous changes.

Time is assumed to pass linearly during a continuous evolution and do not evolves with a discrete change. It is the same assumption as in $d\mathcal{L}$.

Differences with our approach The methodology is similar. The parallel composition of discrete parts is their interleaving via the non-deterministic choice. They define a special parallel composition for continuous components and integrate it with the discrete case.

There are two main differences. Discrete changes have a priority in their system and it restricts the expressiveness of their approach. The second difference is that the parallel continuous composition is linear.

Definition 26. *Linear parallel composition The* linear parallel composition *of differential actions $e_1 : \dot{X} = F(X)$ and $e_2 : \dot{X} = G(X)$ result in the following action:*

$$e_1 \wedge e_2 : \dot{X} = F(X) + G(X) \parallel e_1 \wedge \neg e_2 : \dot{X} = F(X) \parallel \neg e_1 \wedge e_2 : \dot{X} = G(X)$$

The linear parallel composition is associative. But it induces an exponentially increasing number of system of differential equation. It goes against the ideal of scalability. Plus, to our knowledge, most of the CPS of interest share the same evolution domain. They have made the choice of linear parallel composition to implement their approach in HyTech [65].

3.2.4 Relation to the meta-theory of Benveniste

In this section, we discuss how our contributions so far relate to the meta-theory of contract developed by Benveniste *et al.* [18].

Component The authors define a *component* as an open system with typed inputs and which generates outputs. Our definition of a component (Def. 19) is a reactive open system under the form of a hybrid program. The inputs are a part of the free variables of the hybrid program. They are not typed since every variable in $d\mathcal{L}$ is valuated in $\mathbb{R}$. The outputs are defined by the bound variables of the hybrid program.

Environment In the meta-theory, the *environment* of a component corresponds to other components and the exterior world, *i.e.* variables that are not outputs of a component. In our work, the environment is only the exterior world. We dissociate the other components to clarify our approach.

Contract The authors define a *contract* as a pair composed of a subset of components and a subset of environment. It is a very abstract definition. The first subset is the one of components that implement the contract. The second subset represents the environments in which the contract can operate. When it comes to concrete contract-based design theories, they state two important properties. The contract needs to have a finite description that does not directly refer to the actual components. The implementation relation needs to be effectively computable.

In our work, a contract is defined by a pair of formulas (A, G). It is thus a finite description that does not refer to the component which is under the form of a hybrid program α. The implementation relation corresponds to the validity of the formula $A \rightarrow [\alpha]G$. It is indeed computable since there is syntactic rules to decide it.

Composition of components The authors define a *composition operator* to support horizontal processes. It must satisfies two important algebraic properties: commutativity and associativity. The operator we have defined in the Definition 22 exhibit theses two properties.

The authors define also a composability criterion. It must be a syntactic property on pairs of component that defines conditions under which the two components can interact. The two components must not share common outputs. It is one of the condition presented in the next section to be allowed to derive the satisfiability of the conjunction of contracts by the composition of components. They define also a compatibility criterion on contracts which is that there exists an environment in which the two contracts properly interact. It corresponds to the conditions (b) and (c) of the theorem of the next section.

The authors have presented a meta-theory for contracts listing the right properties that a contract-based theory must exhibit. Our component-based approach in $d\mathcal{L}$ matches the stated characteristics. They present also two other contract operators: *contract refinement* and *contract conjunction*. We think that it is possible to extend our approach to instantiate theses constructs.

Conclusion

After having presented the definition of a component in $d\mathcal{L}$ in Section 3.1, we have defined a parallel composition operator to build a system from its parts. The operator is associative and commutative, which is mandatory to modularly design a system. It is also syntactical, making it amenable to automation.

We want to prove that the system resulting from the composition of component is correct. Proving correctness of each individual component is not enough. We have to be able to transfer the correctness of the component to the global system. In the next section, we state and prove the theorem which allow to transfer the proof of correctness through composition. It provides a proof that the result of the composition of two components satisfies the contract $(A_\alpha \wedge A_\beta, G_\alpha \wedge G_\beta)$ provided that each component satisfies their respective contracts (A_α, G_α) and (G_α, G_β).

3.3 Modular proof

We want to transfer the proof of correctness of component through composition. More precisely, let `A` and `B` be components with behaviors α and β and contracts (A_α, G_α) and (A_β, G_β). If `A` and `B` satisfy their respective contracts, then we want that the component `AB` resulting from the composition of `A` and `B` satisfies the contract $(A_\alpha \wedge A_\beta, G_\alpha \wedge G_\beta)$. In $d\mathcal{L}$, it means that if we have a proof tree for the sequents $\mathcal{E}, A_\alpha \vdash [\alpha]G_\alpha$ and $\mathcal{E}, A_\beta \vdash [\beta]G_\beta$, then we want to automatically derive a proof tree for the sequent $\mathcal{E}, A_\alpha, A_\beta \vdash [\alpha \circ \beta](G_\alpha \wedge G_\beta)$.

We present in Subsection 3.3.1 the necessary syntactic conditions to prove the desired theorem. We prove a technical result in Subsection 3.3.2. We state and prove in Subsection 3.3.3 the theorem for the parallel continuous composition operator $\circ_c$. In Subsection 3.3.4, we consider the case where both components are discrete, then in Subsection 3.3.5, the case where one component is discrete and the second one is continuous. Last, we consider the case where both components are general in Subsection 3.3.6.

3.3.1 Necessary conditions

There is three necessary conditions that components must respect to transfer the proof of correctness through composition. They must not share the same outputs. The guarantees of a component must not refer to the output of the other component. The assumptions of a component must be implied by the relevant guarantees of the other component.

No common outputs The outputs of each component must be dissociated. Otherwise, it means that two components `A` and `B` intervene on the same output, and thus have same the same purpose.

In $d\mathcal{L}$, it means that the bound variables of each behaviors are distinct, *i.e.* $BV(\alpha) \cap BV(\beta) = \varnothing$; the behaviors α and β cannot modify the same variable. Otherwise, the execution of $\alpha \circ \beta$ may make no sense and the sequent $\mathcal{E}, A_\alpha, A_\beta \vdash [\alpha \circ \beta](G_\alpha \wedge G_\beta)$ may also be invalid although $\mathcal{E}, A_\alpha \vdash [\alpha]G_\alpha$ and $\mathcal{E}, A_\beta \vdash [\beta]G_\beta$ are valid.

Example 28 (Composition with same outputs). *Let $\alpha = (x := 10)^*$ and $\beta = (x := 100)^*$ be two hybrid programs. We can easily prove $[\alpha](x \leqslant 15)$ and $[\beta](x \leqslant 100)$, but not $[\alpha \circ \beta](x \leqslant 15 \wedge x \leqslant 100)$ which is equivalent to $[(x := 10 \cup x := 100)^*](x \leqslant 15 \wedge x \leqslant 100)$. As soon as the last loop iteration assigns 100 to x, the left-handside $(x \leqslant 15)$ of the formula does not hold anymore.*

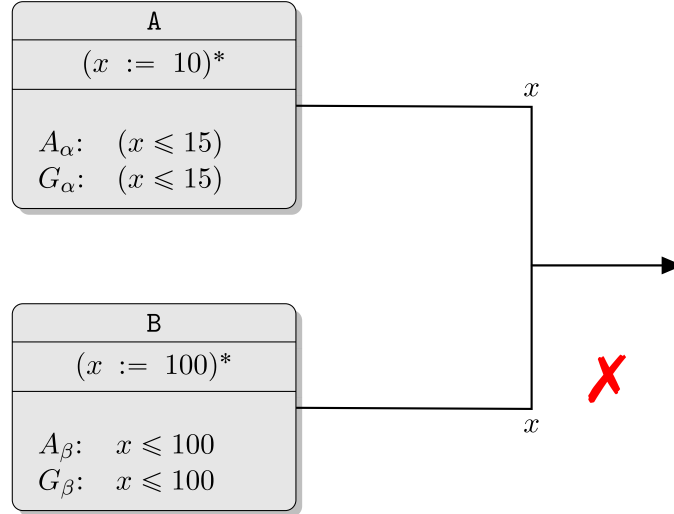


Figure 3.11: Unsafe composition with common outputs

This example provides us with a first necessary condition toward automated derivation of proof trees.

Guarantees do not refer to exterior outputs The second condition requires that the guarantees of the component `A` do not refer to outputs of the component `B`, *i.e.* it does not depend of the behavior of an other component. The converse is also true. It would be aberrant for a tachymeter to refer to the fuel's level of a vehicle.

In $d\mathcal{L}$, it means that variables occurring in G_α (resp. G_β) must not be bound in β (resp. α). The condition amounts to:

$$Var(G_\alpha) \cap BV(\beta) = \varnothing \text{ and } Var(G_\beta) \cap BV(\alpha) = \varnothing$$

Example 29 (Guarantee dependent of another component). *Let* $\alpha = (x := 10)^*$ *and* $\beta = (y := 100)^*$ *be two hybrid programs. Assume that* G_β *refers to bound variables of* α*, e.g. if* $G_\beta \triangleq x \geqslant 42$*. It is easy to prove that it is an invariant of* β*, i.e.* $x \geqslant 42 \vdash [(y := 100)^*]x \geqslant 42$ *is valid. But it is not an invariant of the component* $\alpha \circ \beta$*; the sequent* $x \geqslant 42 \vdash [(x := 10; y := 100)^*]x \geqslant 42$ *is not valid. As soon as the last loop iteration assigns* 10 *to* x*, the formula* $x \geqslant 42$ *does not hold anymore.*

It is graphically represented in the Figure 3.12. A correct guarantee would refer to the output y*.*

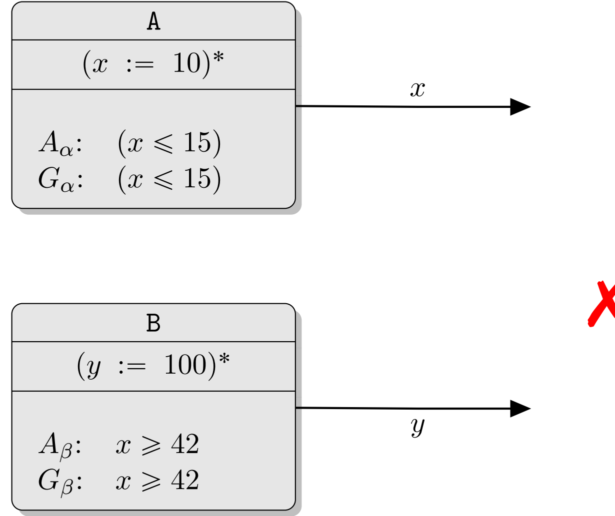


Figure 3.12: Unsafe composition with a guarantee referring to an exterior output

For the continuous case, we request also that the evolution domain of the discrete fragment of a component does not refer to outputs of the other continuous fragment. We request that $Var(H_X) \cap BV(\mathbf{cont}_\beta)$ and $Var(H_Y) \cap BV(\mathbf{cont}_\alpha)$.

Assumptions are not falsified by exterior components A last condition is that the guarantees G_α of a component `A` must implies the assumptions A_β of the component `B` that refers to outputs of `A`, and conversely. It means that the execution of `A` do not break the assumptions under which `B` operated.

In $d\mathcal{L}$, it is translated by the condition that the sequents $A_\beta \vdash \forall^\alpha(G_\alpha \rightarrow A_\beta)$ and $A_\alpha \vdash \forall^\beta(G_\beta \rightarrow A_\alpha)$ are valid.

Example 30 (Assumption not implied by other guarantee). *Let* $\alpha = (x := 10)^*$ *and* $\beta = (y := x)^*$ *be two behaviors,* $G_\alpha \triangleq \top$*,* $A_\beta \triangleq x = 4$ *and* $G_\beta \triangleq y = 4$*. The sequents* $\vdash [\alpha]\top$ *and* $x = 4 \vdash [\beta]y \geqslant 4$ *are valid. Yet, the sequent* $x = 4 \vdash [(x := 10 \cup y := x)^*]y = 4$ *is clearly not valid. A first iteration assigns* 10 *to the variable* x*, then a second assigns* x *to* y*. We have then* $y = x = 10$ *which contradicts* $y = 4$*.*

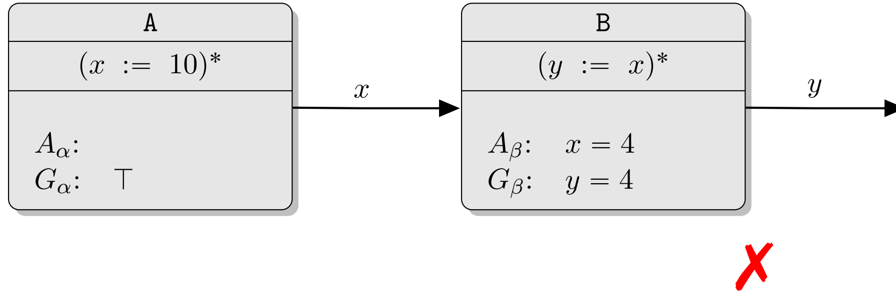


Figure 3.13: Guarantee does not imply the assumption

Conclusion We have presented three necessary conditions for the sound composition of components. They amount to good modeling practices and are mandatory to design a meaningful system.

We believe that theses conditions may be weakened if we develop a more complex model of communications such as in Hybrid Communicating Sequential Processes [73]. HCSP processes uses outputs $ch!e$ to send a message e along a channel ch and inputs $ch?x$ to receive a message along the channels ch. Such channels are unique and thus to an output corresponds precisely an input. It allows a more subtle management of inputs and outputs during the composition. We detail some preliminary results in the future works Section 5.1.3.

We present in the next section a technical result needed to achieve the proofs in Subsection 3.3.3 and 3.3.6. They are related to the conditions we have presented.

3.3.2 Technical result

We state and demonstrate an useful technical result, the *separation lemma*, to achieve the proof of Theorems 1, 2, 3 and 4. It is to make use of the condition that the guarantee of a component must not refer to outputs of another component. We recall the definition of the skolemization of formulas, a notion needed in next subsections.

Definition 27 (Skolemization)**.** *We denote by* $\forall^\alpha\varphi$ *the* skolemization *of a formula with respect to the bound variables of its hybrid program* α. *(We will use the shorter notation* φ^α *where unambiguous).*

Example 31. *Recall that* $BV(\textbf{\textit{Engine}}) = \{s_{eng}\}$. *The skolemization of* G_{tach} *by the behavior* ***Engine*** *is* $G_{tach}^{eng} \triangleq \forall s_{eng}, G_{tach}$.

Separation lemma We need a lemma which state that if the guarantees G_α of a component `A` do not refer to the outputs of the other component `B`, *i.e.* they are *separated*, then `B` does not affect G_α.

Lemma 1 (Separation)**.** *Let* α *be a behavior of a component and* φ *a formula such that* $BV(\alpha) \cap Var(\varphi) = \varnothing$. *Then the skolemization* φ^α *is equivalent to* φ, *i.e.* $\varphi^\alpha \leftrightarrow \varphi$.

Proof. We prove the lemma by induction on the structure of the formula φ.

Let φ be an atomic formula, *e.g.* $x \geqslant \theta$. We have to prove that the equivalence $\forall^\alpha x \geqslant \theta \longleftrightarrow x \geqslant \theta$ is valid. We recall that the bound variables of α are separated of the free variables of φ. We first prove that $\forall^\alpha x \geqslant \theta \rightarrow x \geqslant \theta$.

$$\dfrac{\dfrac{}{x \geqslant \theta \vdash x \geqslant \theta}\,\text{ax}}{\forall^{\alpha} x \geqslant \theta \vdash x \geqslant \theta}\,\forall_l$$

We apply the proof rule $\forall_l$ to eliminate the quantification in the left-hand side of the sequent. Since the quantification is over variables that are not present in the atomic formula $x \geqslant \theta$, it leaves it unchanged. We conclude using the axiom rule. We reason similarly for the second implication $x \geqslant \theta \rightarrow \forall^{\alpha} x \geqslant \theta$.

$$\dfrac{\dfrac{}{x \geqslant \theta \vdash x \geqslant \theta}\,\text{ax}}{x \geqslant \theta \vdash \forall^{\alpha} x \geqslant \theta}\,\forall_r$$

We apply the proof rule $\forall_r$, and since the quantification is over variables that are not present in the atomic formula $x \geqslant \theta$, it leaves the right-hand side unchanged. We conclude using the axiom rule.

Let φ be the conjunction $\varphi_1 \wedge \varphi_2$ and we assume that the formula $\varphi_i^{\alpha} \leftrightarrow \varphi_i$ $(i = 1, 2)$ is valid under the assumption that $BV(\alpha) \cap Var(\varphi_i)$ $(i = 1, 2)$ is empty. We have to prove that the formula $\forall^{\alpha}(\varphi_1 \wedge \varphi_2) \longleftrightarrow \varphi_1 \wedge \varphi_2$ is valid assuming that $BV(\alpha) \cap Var(\varphi_1 \wedge \varphi_2)$. By definition of quantification, $\forall^{\alpha}(\varphi_1 \wedge \varphi_2)$ is equivalent to $(\forall^{\alpha}\varphi_1) \wedge (\forall^{\alpha}\varphi_2)$. By induction hypothesis, $\varphi_i^{\alpha} \leftrightarrow \varphi_i$ $(i = 1, 2)$, and thus $(\forall^{\alpha}\varphi_1) \wedge (\forall^{\alpha}\varphi_2) \longleftrightarrow \varphi_1 \wedge \varphi_2$.

Let φ be an universally quantified formula $\forall x, \varphi_0$. Again, we assume that the formula $\forall^{\alpha}\varphi_0 \longleftrightarrow \varphi_0$ is valid under the assumption that the intersection $BV(\alpha) \cap Var(\varphi_0)$ is empty. By definition, $\forall^{\alpha}\forall x\varphi_0$ is equivalent to $\forall x\forall^{\alpha}\varphi_0$. By applying the induction hypothesis, we obtain the formula $\forall x\varphi_0$. □

3.3.3 For two continuous components

We first state the theorem for the composition of components at the textual representation level. We state and prove after the equivalent result with behaviors in $d\mathcal{L}$ 1.

Theorem. *Let* `A` *and* `B` *be two components with continuous behaviors and respective contracts* (A_α, G_α) *and* (A_β, G_β)*. Assume that they both satisfy their contracts. Furthermore, assume that*

- (a) `A.outputs` $\cap$ `B.outputs` $= \varnothing$,
- (b_1) `A.outputs` *is separated from* G_β *and* H_Y*,*
- (b_2) `B.outputs` *is separated from* G_α *and* H_X*,*
- (c_1) G_α *must implies the assumptions* A_β *that refers to outputs of* `A`*,*
- (c_2) G_β *must implies the assumptions* A_α *that refers to outputs of* `B`*.*

Then the component `AB` *resulting from the parallel composition of* `A` *and* `B` *satisfies the conjunction of contracts* $(A_\alpha \wedge A_\beta, G_\alpha \wedge G_\beta)$*.*

The conditions (a), (b) and (c) are detailed in Subsection 3.3.1. The first assumption (a) assumes components to have separate internal variables and requires them to define disjoint output variables (i.e. unique definitions), which essentially amounts to good modeling practice. The second assumption (b_1) (resp. (b_2)) requires the safety property G_α (resp. G_β) to guard the behavior of the system α (resp. β), i.e. its outputs, and of course not βs (resp. α).

It hence seems natural to require its separation with G_β (resp. G_α). The condition (c_1) (resp. (c_2)) requires the assumptions A_β (resp. A_α) to be implied by the guarantees G_α (resp. G_β).

Recall that the behavior α of a component `A` is the form $\alpha \triangleq \dot{X} = \theta_X \;\;\&\;\; H_X$ and that the behavior β of a component `B` is of the form $\beta \triangleq \dot{Y} = \theta_Y \;\;\&\;\; H_Y$.

Theorem 1. *Let α and β be two continuous behaviors of components* `A` *and* `B` *with respective contracts (A_α, G_α) and (A_β, G_β). Assume that we have two proof trees of $\mathcal{E}, A_\alpha \vdash [\alpha]G_\alpha$ and $\mathcal{E}, A_\beta \vdash [\beta]G_\beta$ respectively, where $\mathcal{E}$ is the environment. Furthermore, assume that*

- (a) $BV(\alpha) \cap BV(\beta) = \varnothing$,
- (b_1) $BV(\alpha) \cap FV(G_\beta) = \varnothing$ *and* $BV(\alpha) \cap FV(H_Y) = \varnothing$,
- (b_2) $BV(\beta) \cap FV(G_\alpha) = \varnothing$ *and* $BV(\beta) \cap FV(H_X) = \varnothing$,
- (c_1) $\mathcal{E}, A_\alpha \vdash \forall^\beta (G_\beta \to A_\alpha)$,
- (c_2) $\mathcal{E}, A_\beta \vdash \forall^\alpha (G_\alpha \to A_\beta)$.

Then it exists a proof tree of $\mathcal{E}, A_\alpha, A_\beta \vdash [\alpha \circ_c \beta](G_\alpha \wedge G_\beta)$.

Proof. Let α and β be two continuous behaviors of components `A` and `B`, and assume that we have a proof tree P_α of $\mathcal{E}, A_\alpha \vdash [\alpha]G_\alpha$ and a proof tree P_β of $\mathcal{E}, A_\beta \vdash [\beta]G_\beta$. We have $\alpha \triangleq \dot{X} = \theta_X \;\;\&\;\; H_X$ and $\beta \triangleq \dot{Y} = \theta_Y \;\;\&\;\; H_Y$.

The composition is still a continuous component and we want a proof tree of $\mathcal{E}, A_\alpha, A_\beta \vdash [\alpha \circ_c \beta](G_\alpha \wedge G_\beta)$ by using the proofs P_α and P_β. We inspect all the rules that can be applied in P_α to prove the sequent $\mathcal{E}, A_\alpha \vdash [\alpha]G_\alpha$ (resp. for the proof tree P_β of $\mathcal{E}, A_\beta \vdash [\beta]G_\beta$) and for each particular association, we give a proof of $\mathcal{E}, A_\alpha, A_\beta \vdash [\alpha \circ_c \beta](G_\alpha \wedge G_\beta)$.

Inspection of proof rules We can apply the following rules to the sequent $\mathcal{E}, A_\alpha \vdash [\alpha]G_\alpha$: differential invariant [DI], ODE solution [ODEsolve], differential weakening [DW], differential cut [DC], Cut and generalization [Gen]. The rules cut, generalization and differential cut do not decompose the ODE since we still have to prove $[\alpha]G_\alpha$ in one of the premises. For example, if we apply the rule Cut, we have the following proof tree:

$$\dfrac{\mathcal{E}, A_\alpha \vdash [\alpha]G_\alpha, \varphi \qquad \mathcal{E}, A_\alpha, \varphi \vdash [\alpha]G_\alpha}{\mathcal{E}, A_\alpha \vdash [\alpha]G_\alpha}\ \text{Cut}$$

We still have to prove the modality $[\alpha]G_\alpha$ in the right premise.

This careful inspection shows that there are only three rules to consider: [DI], [ODEsolve] and [DW]. We first consider the case where $\mathcal{E}, A_\alpha \vdash [\alpha]G_\alpha$ and $\mathcal{E}, A_\beta \vdash [\beta]G_\beta$ are proved by the rule [DI]. For the rule [ODEsolve], we show how to construct a proof using [DI] rule instead. Last, we consider the case where one have been proved by rule [DI] and the other by rule [DW] and the case where both sequents have been proved by rule [DW].

We consider four cases :

- [DI]-[DI] Where the two have been proved using the rule [DI]: we apply also the rule [DI] to the sequent representing the contract of the composite. It is divided in three sub-cases:

- Where both ODEs are independent, *i.e.* the output of one component is not an input of the other
- Where one ODE refers to the other, but the converse is false, *i.e.* the input of one component is an input of the other
- Where both ODEs refer to the other.

The first sub-case basically means that they do not interfere, the proof is thus simple. For the second case, we use the fact that the guarantee of a component implies the assumptions of the other to guarantee a safe composition. In the last sub-case, we show that we get back to the second sub-case.

- [ODEsolve] The case where one component have been proved by the rule [ODEsolve]: we use a result of A. Platzer stating that if we have a proof with the rule [ODEsolve], then we have a proof with the differential induction rule [DI]. We can thus get back to the first case.

- [DW]-[DW] Where both proofs have been achieved by the use of the differential weakening rule: we apply the rule [DW] to the sequent representing the contract of the composite and we retrieve our assumptions easily.

- [DI]-[DW] One sequent have been proved by the differential induction rule and the second by the differential weakening rule: we introduce the guarantee of the first one in the evolution domain with the differential cut rule and apply then the rule [DW]

([DI]-[DI])

If $\mathcal{E}, A_\alpha \vdash [\alpha]G_\alpha$ and $\mathcal{E}, A_\beta \vdash [\beta]G_\beta$ are proved by using rule [DI], then we have a proof tree of this form for P_α:

$$\dfrac{\dfrac{P^{H_X}}{\mathcal{E}, A_\alpha, H_X \vdash G_\alpha} \qquad \dfrac{\dfrac{\dfrac{P'_\alpha}{\mathcal{E}, A_\alpha, H^\alpha_X \vdash (G'^\alpha_\alpha)^{\theta_X}_{\dot{X}}}}{\mathcal{E}, A_\alpha \vdash H^\alpha_X \rightarrow (G'^\alpha_\alpha)^{\theta_X}_{\dot{X}}}\rightarrow_r}{\mathcal{E}, A_\alpha \vdash \forall^\alpha (H_X \rightarrow (G'_\alpha)^{\theta_X}_{\dot{X}})}\forall_r}{\mathcal{E}, A_\alpha \vdash [\dot{X} = \theta_X \ \& \ H_X]G_\alpha}\text{[DI]}$$

The notation $\varphi^{\theta_X}_{\dot{X}}$ stands for the formula φ where all the occurrences of $\dot{X}$ are replaced by θ_X. The left part of the tree is P^{H_X} and the right part P'_α. We can derive a similar proof tree for $\mathcal{E}, A_\beta \vdash [\beta]G_\beta$. We use these sub-parts to close goals of the proof tree of $\mathcal{E}, A_\alpha, A_\beta \vdash [\dot{X}, \dot{Y} = \theta_X, \theta_Y \ \& \ H_X \wedge H_Y](G_\alpha \wedge G_\beta)$.

We have to prove the sequent $\mathcal{E}, A_\alpha, A_\beta \vdash [\dot{X}, \dot{Y} = \theta_X, \theta_Y \ \& \ H_X \wedge H_Y](G_\alpha \wedge G_\beta)$.

We have to be very careful to the fact that the body of an ODE refers to a derived variable on the other component, *e.g.* if $\alpha \triangleq \dot{x} = y$ and y is derived in β. When we reason individually on α, y is assumed to be constant, but when we compose it became a changing value. We have three possibilities : there is no references, one refers to the other but the converse is false, and they both refers to the other. We consider each possibility separately.

Systems are independent We consider the case where both systems are independent, *e.g.* $\alpha \triangleq \dot{x} = 3t$ and $\beta \triangleq \dot{y} = 42$. Thus we assume here that θ_X (resp. θ_Y) does not refer to outputs of β (resp. α), *i.e.* that $FV(\theta_X) \cap BV(\beta) = \varnothing$ (resp. $FV(\theta_Y) \cap BV(\alpha) = \varnothing$).

We first apply the rule [DI] with $G_\alpha \wedge G_\beta$ as invariant.

$$\frac{\mathcal{E}, A_\alpha, A_\beta, H_X, H_Y \vdash G_\alpha \wedge G_\beta \qquad \mathcal{E}, A_\alpha, A_\beta \vdash \forall^{\alpha\beta}\big((H_X \wedge H_Y) \rightarrow (G'_\alpha \wedge G'_\beta)^{\theta_X\ \theta_Y}_{\dot{X}\ \dot{Y}}\big)}{\mathcal{E}, A_\alpha, A_\beta \vdash [\dot{X}, \dot{Y} = \theta_X, \theta_Y \ \ \& \ \ H_X \wedge H_Y](G_\alpha \wedge G_\beta)} \text{[DI]}$$

The left premise correspond to the initial step. We apply the rule $\wedge_r$ to split the goal and end up with the sequent $\mathcal{E}, A_\alpha, A_\beta, H_X, H_Y \vdash G_\alpha$ and $\mathcal{E}, A_\alpha, A_\beta, H_X, H_Y \vdash G_\beta$. By assumption, we have the proof tree P^{H_X} and P^{H_Y} to conclude.

$$\frac{\dfrac{P^{H_X}}{\mathcal{E}, A_\alpha, A_\beta, H_X, H_Y \vdash G_\alpha} \qquad \dfrac{P^{H_Y}}{\mathcal{E}, A_\alpha, A_\beta, H_X, H_Y \vdash G_\beta}}{\mathcal{E}, A_\alpha, A_\beta, H_X, H_Y \vdash G_\alpha \wedge G_\beta} \wedge_r$$

The second premise is the differential induction step. We have to prove the sequent $\mathcal{E}, A_\alpha, A_\beta \vdash \forall^{\alpha\beta}\big((H_X \wedge H_Y) \rightarrow (G'_\alpha \wedge G'_\beta)^{\theta_X\ \theta_Y}_{\dot{X}\ \dot{Y}}\big)$ to finish our proof tree.

Thanks to the condition (a) and (b), the formula $(G'_\alpha \wedge G'_\beta)^{\theta_X\ \theta_Y}_{\dot{X}\ \dot{Y}}$ is equivalent to the formula $(G'_\alpha)^{\theta_X}_{\dot{X}} \wedge (G'_\beta)^{\theta_Y}_{\dot{Y}}$. Since the outputs of each component are dissociated and G_α does not contain occurrences of bound variables of β, *i.e.* of Y, the substitution of $\dot{Y}$ by θ_Y does not have any effect on G_α. Idem for G_β.

We apply the rule $\forall_r$ to eliminate the universal quantification in the goal. By applying the separation lemma (Lemma 1) to the evolution domain $H_X \wedge H_Y$, the resulting sequent is $\mathcal{E}, A_\alpha, A_\beta \vdash (H^\alpha_X \wedge H^\beta_Y) \rightarrow ((G'_\alpha)^{\theta_X}_{\dot{X}} \wedge (G'_\beta)^{\theta_Y}_{\dot{Y}})^{\alpha\beta}$ where the superscript α (resp. β) in G'^α_α means that the bound variables of α (resp. β) occurring in G'_α are replaced by fresh variables. The conditions $(b1)$ and $(b2)$ allows to apply the separation lemma to $(G'_\alpha)^{\theta_X}_{\dot{X}}$ and $(G'_\beta)^{\theta_Y}_{\dot{Y}}$. Thus, we have the sequent $\mathcal{E}, A_\alpha, A_\beta \vdash (H^\alpha_X \wedge H^\beta_Y) \rightarrow (((G'_\alpha)^{\theta_X\ \theta_Y}_{\dot{X}\ \dot{Y}})^\alpha \wedge ((G'_\beta)^{\theta_X\ \theta_Y}_{\dot{X}\ \dot{Y}})^\beta)$.

$$\frac{\mathcal{E}, A_\alpha, A_\beta \vdash (H^\alpha_X \wedge H^\beta_Y) \rightarrow (((G'_\alpha)^{\theta_X}_{\dot{X}})^\alpha \wedge ((G'_\beta)^{\theta_X}_{\dot{X}})^\beta)}{\mathcal{E}, A_\alpha, A_\beta \vdash \forall^{\alpha\beta}\big((H_X \wedge H_Y) \rightarrow (G'_\alpha \wedge G'_\beta)^{\theta_X\ \theta_Y}_{\dot{X}\ \dot{Y}}\big)} \forall_r$$

We apply successively the rules $\rightarrow_r$ and $\wedge_l$ to pass the left-hand side of the implication on the hypothesis.

$$\frac{\dfrac{\mathcal{E}, A_\alpha, A_\beta, H^\alpha_X, H^\beta_Y \vdash ((G'_\alpha)^{\theta_X}_{\dot{X}})^\alpha \wedge ((G'_\beta)^{\theta_X}_{\dot{X}})^\beta}{\mathcal{E}, A_\alpha, A_\beta, H^\alpha_X \wedge H^\beta_Y \vdash ((G'_\alpha)^{\theta_X}_{\dot{X}})^\alpha \wedge ((G'_\beta)^{\theta_X}_{\dot{X}})^\beta} \wedge_l}{\mathcal{E}, A_\alpha, A_\beta \vdash (H^\alpha_X \wedge H^\beta_Y) \rightarrow (((G'_\alpha)^{\theta_X}_{\dot{X}})^\alpha \wedge ((G'_\beta)^{\theta_X}_{\dot{X}})^\beta)} \rightarrow_r$$

We split the conjunction. The left premise is closed by P'_α. the second premise is closed by P'_β.

$$\dfrac{\mathcal{E}, A_\alpha, A_\beta, H_X^{\alpha\beta}, H_Y^\beta \vdash ((G'_\alpha)_{\dot X}^{\theta_X})^{\alpha\beta} \qquad \dfrac{P'_\beta}{\mathcal{E}, A_\alpha, A_\beta, H_X^{\alpha\beta}, H_Y^\beta \vdash ((G'_\beta)_{\dot Y}^{\theta_Y})^\beta}}{\mathcal{E}, A_\alpha, A_\beta, H_X^{\alpha\beta}, H_Y^\beta \vdash ((G'_\alpha)_{\dot X}^{\theta_X})^{\alpha\beta} \wedge ((G'_\beta)_{\dot Y}^{\theta_Y})^\beta} \wedge_r$$

For the left goal, either θ_X does not refer initially to bound variables of β, and we can thus conclude by P'_α. Either, occurrences of y have been replaced by $f(z)$ and we can thus conclude by the proof tree $p'_\alpha 0$.

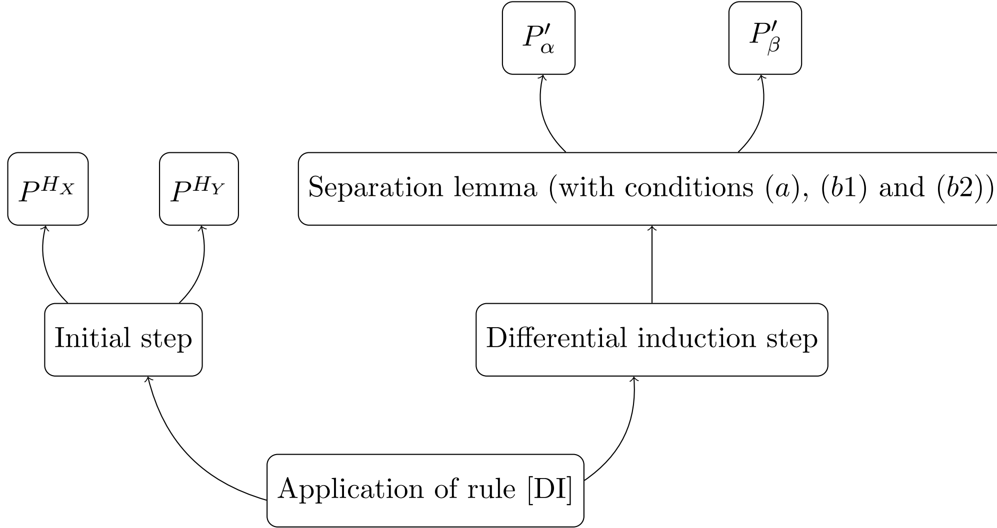


Figure 3.14: Summary of the proof for the case [DI]-[DI]

One system is dependent of the other We assume here that one system is dependent of the other, *e.g.* the speed equations $\dot v = a, \dot x = v$. Here the second part of the equation refers to the derived variable in the first part. More formally, we assume that $FV(\theta_X) \cap BV(\beta) = \{y\}$ and $FV(\theta_Y) \cap BV(\alpha) = \varnothing$. It is easy to generalize it to a set of variables.

If we conduct the proof as in the previous case, we will have trouble with the skolemization step. Indeed, y will be captured and it will not be possible to use A_α to retrieve the proof tree P'_α.

Before applying the differential induction rule [DI], we introduce G_β in the evolution domain thanks to the rule (Cut).

$$\underbrace{\mathcal{E}, A_\alpha, A_\beta \vdash [\dot X, \dot Y = \theta_X, \theta_Y \ \ \& \ \ H_X \wedge H_Y \wedge G_\beta](G_\alpha \wedge G_\beta)}$$

$$\dfrac{\mathcal{E}, A_\alpha, A_\beta \vdash [\dot X, \dot Y = \theta_X, \theta_Y \ \ \& \ \ H_X \wedge H_Y] G_\beta}{\mathcal{E}, A_\alpha, A_\beta \vdash [\dot X, \dot Y = \theta_X, \theta_Y \ \ \& \ \ H_X \wedge H_Y](G_\alpha \wedge G_\beta)} \text{[DC]}$$

The left premise is closed exactly as for the case where the continuous systems are independent. By assumption, α will not interfere and the proof follows without trouble.

We introduce now the formula A_α in the evolution domain of the right premise. The skolemization step will now include A_α and we will be able to retrieve the proof tree P'_α.

$$\underbrace{\mathcal{E}, A_\alpha, A_\beta \vdash [\dot{X}, \dot{Y} = \theta_X, \theta_Y \ \ \& \ \ H_X \wedge H_Y \wedge G_\beta \wedge A_\alpha](G_\alpha \wedge G_\beta)}$$

$$\dfrac{\mathcal{E}, A_\alpha, A_\beta \vdash [\dot{X}, \dot{Y} = \theta_X, \theta_Y \ \ \& \ \ H_X \wedge H_Y \wedge G_\beta]A_\alpha}{\mathcal{E}, A_\alpha, A_\beta \vdash [\dot{X}, \dot{Y} = \theta_X, \theta_Y \ \ \& \ \ H_X \wedge H_Y \wedge G_\beta](G_\alpha \wedge G_\beta)} \text{[DC]}$$

The left premise is closed by an application of the proof rule [DW] and the use of the condition (c).

$$\dfrac{\mathcal{E}, A_\alpha, A_\beta \vdash \forall^{\alpha\beta}\big((H_X \wedge H_Y \wedge G_\beta) \to A_\alpha\big)}{\mathcal{E}, A_\alpha, A_\beta \vdash [\dot{X}, \dot{Y} = \theta_X, \theta_Y \ \ \& \ \ H_X \wedge H_Y \wedge G_\beta]A_\alpha} \text{[DW]}$$

For the right premise, we now apply the differential induction rule [DI] to the sequent $\mathcal{E}, A_\alpha, A_\beta \vdash [\dot{X}, \dot{Y} = \theta_X, \theta_Y \ \ \& \ \ H_X \wedge H_Y \wedge G_\beta \wedge A_\alpha](G_\alpha \wedge G_\beta)$.

$$\underbrace{\mathcal{E}, A_\alpha, A_\beta \vdash \forall^{\alpha\beta}\big((H_X \wedge H_Y \wedge G_\beta \wedge A_\alpha) \to (G'_\alpha \wedge G'_\beta)^{\theta_X\ \theta_Y}_{\dot{X}\ \dot{Y}}\big)}$$

$$\dfrac{\mathcal{E}, A_\alpha, A_\beta \vdash (H_X \wedge H_Y \wedge G_\beta \wedge A_\alpha) \to (G_\alpha \wedge G_\beta)}{\mathcal{E}, A_\alpha, A_\beta \vdash [\dot{X}, \dot{Y} = \theta_X, \theta_Y \ \ \& \ \ H_X \wedge H_Y \wedge G_\beta \wedge A_\alpha](G_\alpha \wedge G_\beta)} \text{[DI]}$$

We now apply the skolemization step as in the previous case and we can conclude.

Both systems are dependent of the other We consider now the case where both systems are dependent of the other, *e.g.* $\dot{x} = y, \dot{y} = x$. Recall that we assume that our ODEs possess a solution $f(t)$. Assume that $f(t)$ is such solution for the differential equation $\dot{y} = x$. We can then replace y in the differential equation $\dot{x} = y$ by $f(t)$. Notice that $f(t)$ may contain occurrences of x. We have now the following system $\dot{x} = f(t), \dot{y} = x$ and we are in the second situation that have already been considered.

(rule [ODESolve])

We consider the case where $\mathcal{E}, A_\alpha \vdash [\dot{X} = \theta_X \ \ \& \ \ H_X]G_\alpha$ is proved by the use of the rule [ODEsolve], which means that we find an explicit solution w.r.t. time to the ODE and we replace each occurrence of X in G_α by this solution. We show that, given a proof using the rule [ODEsolve], we can derive a proof using the differential invariant rule [DI]. This leads us back to the previous situation. We do the proof with only one variable, X. It is easy to generalize it to a system of ODEs. Our reasoning is inspired from [98, p. 247]. By hypothesis, we have the following rule for [ODEsolve]

$$\dfrac{\mathcal{E}, A_\alpha \vdash \forall t \geqslant 0(\forall 0 \leqslant \tilde{t} \leqslant t, (H_X)^{y(\tilde{t})}_X) \to (G_\alpha)^{y(t)}_X}{\mathcal{E}, A_\alpha \vdash [\dot{X} = \theta_X \ \ \& \ \ H_X]G_\alpha} \text{[ODEsolve]}$$

$y(t)$ is the solution of the ODE $\dot{X} = \theta_X$.

Let us introduce a fresh variable t to stand for time in the ODE. It will be evaluated after the rule [DI] by using the differential auxiliaries rule [DA].

$$\frac{\mathcal{E}, A_\alpha \vdash [\dot{X} = \theta_X, \dot{t} = 1 \ \ \& \ \ H_X]G_\alpha}{\mathcal{E}, A_\alpha \vdash [\dot{X} = \theta_X \ \ \& \ \ H_X]G_\alpha}\ [\text{DA}]$$

Also, the solution of the ODE shall contain an occurrence of the initial value. To remember it, we use the auxiliary variable rule [IA].

$$\frac{\mathcal{E}, A_\alpha \vdash [X_0 := X][\dot{X} = \theta_X, \dot{t} = 1 \ \ \& \ \ H_X]G_\alpha}{\mathcal{E}, A_\alpha \vdash [\dot{X} = \theta_X, \dot{t} = 1 \ \ \& \ \ H_X]G_\alpha}\ [\text{IA}]$$

The proof tree of $\mathcal{E}, A_\alpha \vdash [\alpha]G_\alpha$ using the rule [DI] is in the Figure 3.3.3. To introduce the solution as an invariant, we use the generalization rule [gen].

$$\cfrac{\cfrac{\Pi_1}{\cfrac{\mathcal{E}, A_\alpha, X^0 = y(t^0) \vdash G_\alpha^0}{\cfrac{\mathcal{E}, A_\alpha \vdash X^0 = y(t^0) \to G_\alpha^0}{\mathcal{E}, A_\alpha \vdash \forall X \forall t, X = y(t) \to G_\alpha}\ \forall_r}\ \to_r}}{}$$

$$\cfrac{\cfrac{\cfrac{}{\mathcal{E}, A_\alpha, H_X \vdash X = y(0)}\ \text{ax} \qquad \cfrac{\cfrac{\cfrac{\cfrac{}{\mathcal{E}, A_\alpha, H_X \vdash \theta_X = \theta_X}\ \text{ax}}{\mathcal{E}, A_\alpha, H_X \vdash (X = y(t))'^{\theta_X\ 1}_{\dot{X}\ \dot{t}}}\ (\text{i})}{\mathcal{E}, A_\alpha \vdash H_X \to (X = y(t))'^{\theta_X\ 1}_{\dot{X}\ \dot{t}}}\ \to_r}{\mathcal{E}, A_\alpha \vdash \forall X, H_X \to (X = y(t))'^{\theta_X\ 1}_{\dot{X}\ \dot{t}}}\ \forall_r}{\mathcal{E}, A_\alpha \vdash [\dot{X} = \theta_X, \dot{t} = 1 \ \ \& \ \ H_X]X = y(t)}\ [\text{DI}]}{\mathcal{E}, A_\alpha \vdash [\dot{X} = \theta_X, \dot{t} = 1 \ \ \& \ \ H_X]G_\alpha}\ [\text{gen}]$$

By hypothesis, we assume a proof P of $\vdash \varphi_X^{y(t^0)}$. Here, t^0 is the fresh variable introduced by the rule $\forall_r$, idem for X^0 and φ^0. On the branch Π_1, we apply exactly the same logical rules that the ones applied in P. For the logical connectives, there is no problem since the structure is similar. We need to pay a little more attention to the case of arithmetic over reals. Indeed, suppose we have proved $y(t^0) \leqslant \theta_X$ in P. Then, we will have to prove $X^0 \leqslant \theta$ in the proof of G_α^0. Fortunately, we also know that $X^0 = y(t)$ by hypothesis. Hence the conclusion by applying basic arithmetic.

For the left premise of rule [DI], we must remember that it is here to prove the initial condition: the property at time $t = 0$, i.e. to ensure that the initials conditions prove the (inductive) safety property. But the value of X is exactly $y(0)$, so we can conclude with the axiom rule. For the right premise, we just perform the substitution and the step (i) is justified by the same reasoning as for [DI].

([DW]-[DW])

We still need to prove the differential weakening case. The proofs are of the following form:

$$\dfrac{\dfrac{\dfrac{\dfrac{P^\alpha}{\mathcal{E}, A_\alpha, H_X^\alpha \vdash G_\alpha^\alpha}}{\mathcal{E}, A_\alpha \vdash H_X^\alpha \to G_\alpha^\alpha} \to_r}{\mathcal{E}, A_\alpha \vdash \forall^\alpha (H_X \to G_\alpha)} \forall_r}{\mathcal{E}, A_\alpha \vdash [\dot{X} = \theta_X \ \ \& \ \ H_X] G_\alpha} \text{[DW]}$$

Once $\mathcal{E}, A_\alpha \vdash [\alpha] G_\alpha$ and $\mathcal{E}, A_\beta \vdash [\beta] G_\beta$ are both proved by the mean of differential weakening, then the proof of the composition is the following:

$$\dfrac{\dfrac{\dfrac{\dfrac{P^\alpha}{\mathcal{E}, A_\alpha, A_\beta, H_X^\alpha, H_Y^\beta \vdash G_\alpha^\alpha} \quad \dfrac{P^\beta}{\mathcal{E}, A_\alpha, A_\beta, H_X^\alpha, H_Y^\beta \vdash G_\beta^\beta}}{\mathcal{E}, A_\alpha, A_\beta \vdash (H_X^\alpha \wedge H_Y^\beta) \to (G_\alpha^\alpha \wedge G_\beta^\beta)} \wedge_r}{\mathcal{E}, A_\alpha, A_\beta \vdash \forall^{\alpha\beta} \big((H_X \wedge H_Y) \to (G_\alpha \wedge G_\beta) \big)} \forall_r}{\mathcal{E}, A_\alpha, A_\beta \vdash [\alpha \circ_c \beta](G_\alpha \wedge G_\beta)} \text{[DW]}$$

It is a very simple case, where the evolution domains are sufficient to prove the safety properties. We first apply the differential weakening rule [DW]. Then the quantification rule which results in the sequent $\mathcal{E}, A_\alpha, A_\beta \vdash (H_X^\alpha \wedge H_Y^\beta) \to (G_\alpha^\alpha \wedge G_\beta^\beta)$ thanks to the separation lemma and conditions (a), $(b1)$ and $(b2)$. We split then the conjunction and ends up with premises $\mathcal{E}, A_\alpha, A_\beta, H_X^\alpha, H_Y^\beta \vdash G_\alpha^\alpha$ and $\mathcal{E}, A_\alpha, A_\beta, H_X^\alpha, H_Y^\beta \vdash G_\beta^\beta$. They are closed by the proof tree P^α and P^β.

([DI]-[DW])

Assume that $\mathcal{E}, A_\alpha \vdash [\alpha] G_\alpha$ have been proved by the use of rule [DI] and $\mathcal{E}, A_\beta \vdash [\beta] G_\beta$ by rule [DW]. The idea is to add G_α into the evolution domain of $\alpha \circ_c \beta$ using the differential cut rule [DC].

$$\overbrace{\mathcal{E}, A_\alpha, A_\beta \vdash [\dot{X}, \dot{Y} = \theta_X, \theta_Y \ \ \& \ \ H_X \wedge H_Y \wedge G_\alpha](G_\alpha \wedge G_\beta)}$$

$$\dfrac{\mathcal{E}, A_\alpha, A_\beta \vdash [\dot{X}, \dot{Y} = \theta_X, \theta_Y \ \ \& \ \ H_X \wedge H_Y] G_\alpha}{\mathcal{E}, A_\alpha, A_\beta \vdash [\dot{X}, \dot{Y} = \theta_X, \theta_Y \ \ \& \ \ H_X \wedge H_Y](G_\alpha \wedge G_\beta)} \text{[DC]}$$

The left premise requires to prove that G_α is an invariant of $\alpha \circ_c \beta$. We can follow the methodology for the case where both components have been proved by differential induction with $\top$ as invariant of G_β. The right premise requires to prove that $G_\alpha \wedge G_\beta$ is an invariant of $\alpha \circ_c \beta$. We apply the differential weakening rule [DW].

$$\dfrac{\mathcal{E}, A_\alpha, A_\beta \vdash \forall^{\alpha\beta} \big((H_X \wedge H_Y \wedge G_\alpha) \to (G_\alpha \wedge G_\beta) \big)}{\mathcal{E}, A_\alpha, A_\beta \vdash [\dot{X}, \dot{Y} = \theta_X, \theta_Y \ \ \& \ \ H_X \wedge H_Y \wedge G_\alpha](G_\alpha \wedge G_\beta)} \text{[DW]}$$

We apply the skolemization step with the rule $\forall_r$ and, thanks to the separation lemma (cf Lemma 1) with conditions (a), $(b1)$ and $(b2)$, we obtain the sequent $\mathcal{E}, A_\alpha, A_\beta \vdash (H_X^\alpha \wedge H_Y^\beta \wedge G_\alpha^\alpha) \rightarrow (G_\alpha^\alpha \wedge G_\beta^\beta)$.

$$\dfrac{\mathcal{E}, A_\alpha, A_\beta \vdash (H_X^\alpha \wedge H_Y^\beta \wedge G_\alpha^\alpha) \rightarrow (G_\alpha^\alpha \wedge G_\beta^\beta)}{\mathcal{E}, A_\alpha, A_\beta \vdash \forall^{\alpha\beta}\big((H_X \wedge H_Y \wedge G_\alpha) \rightarrow (G_\alpha \wedge G_\beta)\big)}\ \forall_r$$

We apply successively the rules $\rightarrow_r$, $\wedge_l$ and $\wedge_r$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathcal{E}, A_\alpha, A_\beta, H_X^\alpha, H_Y^\beta, G_\alpha^\alpha \vdash G_\alpha^\alpha}\ \text{ax} \quad \dfrac{P^\beta}{\mathcal{E}, A_\alpha, A_\beta, H_X^\alpha, H_Y^\beta, G_\alpha^\alpha \vdash G_\beta^\beta}}{\mathcal{E}, A_\alpha, A_\beta, H_X^\alpha, H_Y^\beta, G_\alpha^\alpha \vdash G_\alpha^\alpha \wedge G_\beta^\beta}\ \wedge_r}{\mathcal{E}, A_\alpha, A_\beta, H_X^\alpha, H_Y^\beta \wedge G_\alpha^\alpha \vdash G_\alpha^\alpha \wedge G_\beta^\beta}\ \wedge_l}{\mathcal{E}, A_\alpha, A_\beta, H_X^\alpha \wedge H_Y^\beta \wedge G_\alpha^\alpha \vdash G_\alpha^\alpha \wedge G_\beta^\beta}\ \wedge_l}{\mathcal{E}, A_\alpha, A_\beta \vdash (H_X^\alpha \wedge H_Y^\beta \wedge G_\alpha^\alpha) \rightarrow (G_\alpha^\alpha \wedge G_\beta^\beta)}\ \rightarrow_r$$

The left branch is closed by the rule ax. The right sequent is closed by P^β as in the case of [DW]-[DW].

□

When we compose two components with continuous behaviors, the Theorem 1 ensures that the resulting component satisfies the conjunction of contracts. We have a similar result for the composition of purely discrete components in Subsection 3.3.4.

3.3.4 For two discrete components

As in the previous subsection, we have a theorem ensuring that we retain contracts through parallel composition of purely discrete components. The conditions are similar to the Theorem 1.

Theorem. *Let* `A` *and* `B` *be two discrete components with respective contracts* (A_α, G_α) *and* (A_β, G_β). *Assume that they both satisfy their contracts. Furthermore, assume that*

- (a) `A.outputs` $\cap$ `B.outputs` $= \varnothing$,
- (b_1) `A.outputs` *is separated from* G_β
- (b_2) `B.outputs` *is separated from* G_α,
- (c_1) G_α *must implies the assumptions* A_β *that refer to outputs of* `A`
- (c_2) G_β *must implies the assumptions* A_α *that refer to outputs of* `B`.

Then the component `AB` *resulting from the parallel composition of* `A` *and* `B` *satisfies the conjunction of contracts* $(A_\alpha \wedge A_\beta, G_\alpha \wedge G_\beta)$.

We have the equivalent theorem with the behaviors in $d\mathcal{L}$.

Theorem 2. *Let α and β be two discrete behaviors of components* A *and* B *with respective contracts (A_α, G_α) and (A_β, G_β). Assume that we have two proof trees of $\mathcal{E}, A_\alpha \vdash [\alpha]G_\alpha$ and $\mathcal{E}, A_\beta \vdash [\beta]G_\beta$ respectively, where $\mathcal{E}$ is the environment. Furthermore, assume that*

(a) $BV(\alpha) \cap BV(\beta) = \varnothing$,

(b_1) $BV(\alpha) \cap FV(G_\beta) = \varnothing$,

(b_2) $BV(\beta) \cap FV(G_\alpha) = \varnothing$,

(c_1) $\mathcal{E}, A_\alpha \vdash \forall^\beta(G_\beta \rightarrow A_\alpha)$,

(c_2) $\mathcal{E}, A_\beta \vdash \forall^\alpha(G_\alpha \rightarrow A_\beta)$.

Then it exists a proof tree of $\mathcal{E}, A_\alpha, A_\beta \vdash [\alpha \circ \beta](G_\alpha \wedge G_\beta)$.

The conditions are similar to the Theorem 1, except for conditions ($b1$) and ($b2$) where we do not refer to the evolution domain as it is non-existent since we are working with purely discrete components.

Proof. Assume $\alpha \triangleq [(\mathbf{disc}_\alpha)^*]$ and $\beta \triangleq [(\mathbf{disc}_\beta)^*]$ are two behaviors of component A and B respecting the conditions above. The composition $\alpha \circ \beta$ is given by the behavior $(\mathbf{disc}_\alpha \cup \mathbf{disc}_\beta)^*$. Given the proof of contracts $\mathcal{E}, A_\alpha \vdash [\alpha]G_\alpha$ and $\mathcal{E}, A_\beta \vdash [\beta]G_\beta$, we want to prove the sequent $\mathcal{E}, A_\alpha, A_\beta \vdash [\alpha \circ \beta](G_\alpha \wedge G_\beta)$.

An application of the rule [Ind] have been necessary to prove the sequent $\mathcal{E}, A_\alpha \vdash [(\mathbf{disc}_\alpha)^*]G_\alpha$. It results in two premises : the initial step and the induction step. We assume that we have a proof tree P_{init_α} such that the premise corresponding to the initial step is closed by P_{init_α}. For the second premise, we apply the proof rule $\forall_r$ to skolemize bound variables of α. We recall that the superscript $^\alpha$ on a formula or a hybrid program means that we have replaced every occurrences of bound variables of α by fresh variables. The remaining goal $\mathcal{E}, A_\alpha, G^\alpha_\alpha \vdash [\mathbf{disc}^\alpha_\alpha]G^\alpha_\alpha$ is closed by the proof tree P_{ind_α}.

$$\dfrac{\dfrac{P_{init_\alpha}}{\mathcal{E}, A_\alpha \vdash G_\alpha} \qquad \dfrac{\dfrac{\dfrac{P_{ind_\alpha}}{\mathcal{E}, A_\alpha, G^\alpha_\alpha \vdash [\mathbf{disc}^\alpha_\alpha]G^\alpha_\alpha}}{\mathcal{E}, A_\alpha \vdash G^\alpha_\alpha \rightarrow [\mathbf{disc}^\alpha_\alpha]G^\alpha_\alpha}\rightarrow_r}{\mathcal{E}, A_\alpha \vdash \forall^\alpha(G_\alpha \rightarrow [\mathbf{disc}_\alpha]G_\alpha)}\forall_r}{\mathcal{E}, A_\alpha \vdash [(\mathbf{disc}_\alpha)^*]G_\alpha}[\text{Ind}]$$

We apply a similar reasoning to the proof of the sequent $\beta \triangleq [(\mathbf{disc}_\beta)^*]$. We have thus the proof tree P_{init_β} and P_{ind_β}.

To prove the sequent $\mathcal{E}, A_\alpha, A_\beta \vdash [(\mathbf{disc}_\alpha \cup \mathbf{disc}_\beta)^*](G_\alpha \wedge G_\beta)$, we apply the rule [Ind].

$$\dfrac{\mathcal{E}, A_\alpha, A_\beta \vdash G_\alpha \wedge G_\beta \qquad \mathcal{E}, A_\alpha, A_\beta \vdash \forall^{\alpha\beta}\big((G_\alpha \wedge G_\beta) \rightarrow [\mathbf{disc}_\alpha \cup \mathbf{disc}_\beta](G_\alpha \wedge G_\beta)\big)}{\mathcal{E}, A_\alpha, A_\beta \vdash [(\mathbf{disc}_\alpha \cup \mathbf{disc}_\beta)^*](G_\alpha \wedge G_\beta)}[\text{Ind}]$$

The left premise corresponds to the initial step. We have to prove that both guarantees hold after zero executions. We split the goal with the rule $\wedge_r$ and obtain two premises that correspond to the initial step of each individual proof. They are closed by using the proof trees P_{init_α} and P_{init_β}.

$$\dfrac{\dfrac{P_{init_\alpha}}{\mathcal{E}, A_\alpha, A_\beta \vdash G_\alpha} \quad \dfrac{P_{init_\beta}}{\mathcal{E}, A_\alpha, A_\beta \vdash G_\beta}}{\mathcal{E}, A_\alpha, A_\beta \vdash G_\alpha \wedge G_\beta} \wedge_r$$

The right premise corresponds to the induction step. We first apply the skolemization step with the rule $\forall_r$ to the sequent $\mathcal{E}, A_\alpha, A_\beta \vdash \forall^{\alpha\beta}\big((G_\alpha \wedge G_\beta) \rightarrow [\mathbf{disc}_\alpha \cup \mathbf{disc}_\beta](G_\alpha \wedge G_\beta)\big)$.

$$\dfrac{\mathcal{E}, A_\alpha, A_\beta \vdash (G_\alpha^{\alpha\beta} \wedge G_\beta^{\alpha\beta}) \rightarrow [\mathbf{disc}_\alpha^{\alpha\beta} \cup \mathbf{disc}_\beta^{\alpha\beta}](G_\alpha^{\alpha\beta} \wedge G_\beta^{\alpha\beta})}{\mathcal{E}, A_\alpha, A_\beta \vdash \forall^{\alpha\beta}\big((G_\alpha \wedge G_\beta) \rightarrow [\mathbf{disc}_\alpha \cup \mathbf{disc}_\beta](G_\alpha \wedge G_\beta)\big)}$$

We can apply the separation lemma (cf Lemma 1) since the bound variables are dissociated (condition (a)) and the variables of G_α (resp. G_β) are different from the bound variables of β (resp. α) (conditions (b_1) and (b_2)). Thus, the sequent $\mathcal{E}, A_\alpha, A_\beta \vdash (G_\alpha^{\alpha\beta} \wedge G_\beta^{\alpha\beta}) \rightarrow [\mathbf{disc}_\alpha^{\alpha\beta} \cup \mathbf{disc}_\beta^{\alpha\beta}](G_\alpha^{\alpha\beta} \wedge G_\beta^{\alpha\beta})$ is equivalent to $\mathcal{E}, A_\alpha, A_\beta \vdash (G_\alpha^{\alpha} \wedge G_\beta^{\beta}) \rightarrow [\mathbf{disc}_\alpha^{\alpha\beta} \cup \mathbf{disc}_\beta^{\alpha\beta}](G_\alpha^{\alpha} \wedge G_\beta^{\beta})$.

We apply the rule $\rightarrow_r$ to this sequent followed by the rule $\wedge_l$ to pass in the hypothesis $(G_\alpha^{\alpha} \wedge G_\beta^{\beta})$.

$$\dfrac{\dfrac{\mathcal{E}, A_\alpha, A_\beta, G_\alpha^{\alpha}, G_\beta^{\beta} \vdash [\mathbf{disc}_\alpha^{\alpha\beta} \cup \mathbf{disc}_\beta^{\alpha\beta}](G_\alpha^{\alpha} \wedge G_\beta^{\beta})}{\mathcal{E}, A_\alpha, A_\beta, G_\alpha^{\alpha} \wedge G_\beta^{\beta} \vdash [\mathbf{disc}_\alpha^{\alpha\beta} \cup \mathbf{disc}_\beta^{\alpha\beta}](G_\alpha^{\alpha} \wedge G_\beta^{\beta})} \wedge_l}{\mathcal{E}, A_\alpha, A_\beta \vdash (G_\alpha^{\alpha} \wedge G_\beta^{\beta}) \rightarrow [\mathbf{disc}_\alpha^{\alpha\beta} \cup \mathbf{disc}_\beta^{\alpha\beta}](G_\alpha^{\alpha} \wedge G_\beta^{\beta})} \rightarrow_r$$

The next step is to split $\mathbf{disc}_\alpha^{\alpha\beta} \cup \mathbf{disc}_\beta^{\alpha\beta}$ with the non-deterministic rule $[\cup]$ to consider separately their action. We obtain two premises which are similar to handle. We consider only the case of the left premise, *i.e.* the sequent $\mathcal{E}, A_\alpha, A_\beta, G_\alpha^{\alpha}, G_\beta^{\beta} \vdash [\mathbf{disc}_\alpha^{\alpha\beta}](G_\alpha^{\alpha} \wedge G_\beta^{\beta})$, the second one is closed by the same reasoning.

$$\dfrac{\mathcal{E}, A_\alpha, A_\beta, G_\alpha^{\alpha}, G_\beta^{\beta} \vdash [\mathbf{disc}_\alpha^{\alpha\beta}](G_\alpha^{\alpha} \wedge G_\beta^{\beta}) \qquad \mathcal{E}, A_\alpha, A_\beta, G_\alpha^{\alpha}, G_\beta^{\beta} \vdash [\mathbf{disc}_\beta^{\alpha\beta}](G_\alpha^{\alpha} \wedge G_\beta^{\beta})}{\mathcal{E}, A_\alpha, A_\beta, G_\alpha^{\alpha}, G_\beta^{\beta} \vdash [\mathbf{disc}_\alpha^{\alpha\beta} \cup \mathbf{disc}_\beta^{\alpha\beta}](G_\alpha^{\alpha} \wedge G_\beta^{\beta})} [\cup]$$

We have to prove the sequent $\mathcal{E}, A_\alpha, A_\beta, G_\alpha^{\alpha}, G_\beta^{\beta} \vdash [\mathbf{disc}_\alpha^{\alpha\beta}](G_\alpha^{\alpha} \wedge G_\beta^{\beta})$. We split the guarantees by using the rule [BoxAnd] since they are of different nature. G_α^{α} is a guarantee of the component `A` and we have already a proof by assumption. G_β^{β} is a guarantee of the component `B`, but it is not affected by the execution of α thanks to the conditions (a) and (b).

$$\dfrac{\mathcal{E}, A_\alpha, A_\beta, G_\alpha^{\alpha}, G_\beta^{\beta} \vdash [\mathbf{disc}_\alpha^{\alpha\beta}]G_\alpha^{\alpha} \qquad \mathcal{E}, A_\alpha, A_\beta, G_\alpha^{\alpha}, G_\beta^{\beta} \vdash [\mathbf{disc}_\alpha^{\alpha\beta}]G_\beta^{\beta}}{\mathcal{E}, A_\alpha, A_\beta, G_\alpha^{\alpha}, G_\beta^{\beta} \vdash [\mathbf{disc}_\alpha^{\alpha\beta}](G_\alpha^{\alpha} \wedge G_\beta^{\beta})} [\text{BoxAnd}]$$

We have a proof tree P_{ind_α} of $\mathcal{E}, A_\alpha, G_\alpha^{\alpha} \vdash [\mathbf{disc}_\alpha^{\alpha}]G_\alpha^{\alpha}$ and we want to close the left premise : $\mathcal{E}, A_\alpha, A_\beta, G_\alpha^{\alpha}, G_\beta^{\beta} \vdash [\mathbf{disc}_\alpha^{\alpha\beta}]G_\alpha^{\alpha}$. Here $\mathbf{disc}_\alpha^{\alpha}$ differs from $\mathbf{disc}_\alpha^{\alpha\beta}$ by the variables of β that are captured by the skolemization. We can not make use of the assumption A_α to conclude since it may refer to bound variables of β.

The condition (c) states that the sequent $A_\alpha \vdash G_\beta^{\beta} \rightarrow A_\alpha^{\beta}$ is valid. We introduce the formula $G_\beta^{\beta} \rightarrow A_\alpha^{\beta}$ with the cut rule (Cut).

$$\dfrac{\mathcal{E}, A_\alpha, A_\beta, G_\alpha^\alpha, G_\beta^\beta, G_\beta^\beta \rightarrow A_\alpha^\beta \vdash [\mathbf{disc}_\alpha^{\alpha\beta}]G_\alpha^\alpha \qquad \mathcal{E}, A_\alpha, A_\beta, G_\alpha^\alpha, G_\beta^\beta \vdash [\mathbf{disc}_\alpha^{\alpha\beta}]G_\alpha^\alpha, G_\beta^\beta \rightarrow A_\alpha^\beta}{\mathcal{E}, A_\alpha, A_\beta, G_\alpha^\alpha, G_\beta^\beta \vdash [\mathbf{disc}_\alpha^{\alpha\beta}]G_\alpha^\alpha} \text{(Cut)}$$

The right premise is closed by the assumption that we have a proof of $G_\beta^\beta \rightarrow A_\alpha^\beta$. For the left premise, we can deduce the formula A_α^β in the hypothesis thanks to an application of the rule $\rightarrow_l$. The left premise is closed by the rule ax.

$$\dfrac{\dfrac{}{\mathcal{E}, A_\alpha, A_\beta, G_\alpha^\alpha, G_\beta^\beta \vdash G_\beta^\beta, [\mathbf{disc}_\alpha^{\alpha\beta}]G_\alpha^\alpha}\text{ax} \qquad \mathcal{E}, A_\alpha, A_\beta, G_\alpha^\alpha, G_\beta^\beta, A_\alpha^\beta \vdash [\mathbf{disc}_\alpha^{\alpha\beta}]G_\alpha^\alpha}{\mathcal{E}, A_\alpha, A_\beta, G_\alpha^\alpha, G_\beta^\beta, G_\beta^\beta \rightarrow A_\alpha^\beta \vdash [\mathbf{disc}_\alpha^{\alpha\beta}]G_\alpha^\alpha} \rightarrow_l$$

We are left with the right sequent $\mathcal{E}, A_\alpha, A_\beta, G_\alpha^\alpha, G_\beta^\beta, G_\beta^\beta \rightarrow A_\alpha^\beta, A_\alpha^\beta \vdash [\mathbf{disc}_\alpha^{\alpha\beta}]G_\alpha^\alpha$, and by assumption we have a proof P^α of it.

To prove the remaining sequent $\mathcal{E}, A_\alpha, A_\beta, G_\alpha^\alpha, G_\beta^\beta \vdash [\mathbf{disc}_\alpha^{\alpha\beta}]G_\beta^\beta$, we symbolically execute $\mathbf{disc}_\alpha^{\alpha\beta}$. Since G_β^β does not contain any bound variables of α (and thus of $\mathbf{disc}_\alpha^{\alpha\beta}$), it is not affected. We conclude then by the rule *ax*.

$$\dfrac{\dfrac{}{\mathcal{E}, A_\alpha, A_\beta, G_\alpha^\alpha, G_\beta^\beta \vdash G_\beta^\beta}\,ax}{\mathcal{E}, A_\alpha, A_\beta, G_\alpha^\alpha, G_\beta^\beta \vdash [\mathbf{disc}_\alpha^{\alpha\beta}]G_\beta^\beta} \text{Symbolic execution}$$

We have a summary of the structure of the proof tree in the Figure 3.15.

□

3.3.5 For a discrete component and a continuous component

We consider the case where one discrete component `A` is composed with a continuous component `B`. We mix here reasoning for discrete and continuous composition, and thus have to be very careful for the composition. As for the two previous section, we state the theorem for the components expressed with the textual representation and the equivalent results with behaviors.

Theorem. *Let* `A` *be a discrete component and* `B` *be a continuous component with respective contracts* (A_α, G_α) *and* (A_β, G_β)*. Assume that they both satisfy their contracts. Furthermore, assume that*

- (a) `A.outputs` $\cap$ `B.outputs` $= \varnothing$,
- (b_1) `A.outputs` *is separated from* G_β
- (b_2) `B.outputs` *is separated from* G_α,
- (c_1) G_α *must implies the assumptions* A_β *that refer to outputs of* `A`
- (c_2) G_β *must implies the assumptions* A_α *that refer to outputs of* `B`.

Then the component `AB` *resulting from the parallel composition of* `A` *and* `B` *satisfies the conjunction of contracts* $(A_\alpha \wedge A_\beta, G_\alpha \wedge G_\beta)$.

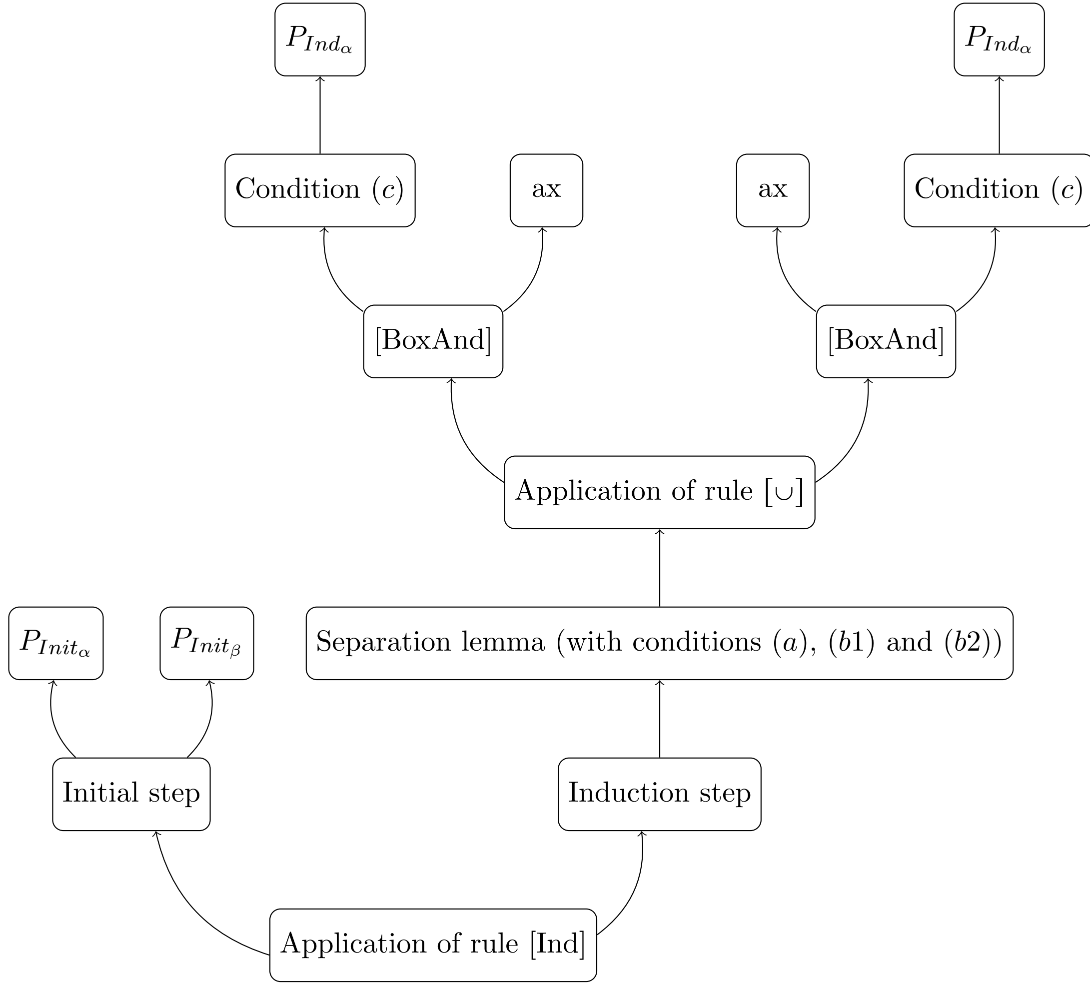


Figure 3.15: Summary of the proof for the discrete case

We have the equivalent theorem for behaviors of components. Recall that the behavior β of a component `B` is of the form $\beta \triangleq \dot{Y} = \theta_Y \ \& \ H_Y$.

Theorem 3. *Let α be the discrete behaviors of a component* `A` *and β be the continuous behavior of a component* `B` *with respective contracts (A_α, G_α) and (A_β, G_β). Assume that we have two proof trees of $\mathcal{E}, A_\alpha \vdash [\alpha]G_\alpha$ and $\mathcal{E}, A_\beta \vdash [(\beta)^*]G_\beta$ respectively, where $\mathcal{E}$ is the environment. Furthermore, assume that*

- (a) $BV(\alpha) \cap BV(\beta) = \varnothing$,
- (b_1) $BV(\alpha) \cap FV(G_\beta) = \varnothing$ *and* $BV(\alpha) \cap FV(H_Y) = \varnothing$,
- (b_2) $BV(\beta) \cap FV(G_\alpha) = \varnothing$,
- (c_1) $\mathcal{E}, A_\alpha \vdash \forall^\beta (G_\beta \rightarrow A_\alpha)$,
- (c_2) $\mathcal{E}, A_\beta \vdash \forall^\alpha (G_\alpha \rightarrow A_\beta)$.

Then it exists a proof tree of $\mathcal{E}, A_\alpha, A_\beta \vdash [\alpha \circ \beta](G_\alpha \wedge G_\beta)$.

We also require that the bound variables of the discrete behavior do not occur in the evolution domain H_Y of β. It is similar to the condition $(b1)$ of the Theorem 1.

There is a major point to consider in this theorem. Instead of assuming having a proof of the sequent $\mathcal{E}, A_\beta \vdash [\beta]G_\alpha$, we assume having a proof of $\mathcal{E}, A_\beta \vdash [(\beta)^*]G_\alpha$. Indeed, when we compose, the differential equation is now under an iteration. Thus, it may not access to assumptions due to supplementary skolemization. We provide a counter-example to show that without this assumption, the theorem does not hold.

Example 32 (Counter-example)**.** *Assume that the behavior of our continuous component is* $\dot{x} = y \ \& \ x = y$*. Its contract is* $(\top, x \leqslant 10)$ *and the initial value of* x *is* 0*. It is simple to obtain a proof of the sequent* $x = 0 \vdash [\dot{x} = y \ \& \ x = y]x \leqslant 10$*. Indeed, we have that* $\dot{x} = 0$*, and thus* x *does not evolve. Since it is initialized at* 0*, we have that* $x \leqslant 10$*.*

Let compose it with a behavior $\boldsymbol{disc}_\alpha$ *of a discrete component and with contract* $(\top, \top)$*. The behavior resulting from the composition is* $(\boldsymbol{disc}_\alpha \cup \dot{x} = y \ \& \ x = y)^*$*. We want to prove the following sequent:*

$$x = 0 \vdash [(\boldsymbol{disc}_\alpha \cup \dot{x} = y \ \& \ x = y)^*]x \leqslant 10$$

We start by applying the induction rule [Ind] and obtain two sub-goals, one for the initial step, the other for the induction step.

$$\dfrac{x = 0 \vdash x \leqslant 10 \qquad x = 0 \vdash \forall x, (x \leqslant 10 \rightarrow [\boldsymbol{disc}_\alpha \cup \dot{x} = y \ \& \ x = y]x \leqslant 10)}{x = 0 \vdash [(\boldsymbol{disc}_\alpha \cup \dot{x} = y \ \& \ x = y)^*]x \leqslant 10}\ \textit{[Ind]}$$

Consider the induction step. We apply the rule $\forall_r$ *and replace the variable* x *by the fresh variable* x_0*. We then split the behavior between* $\boldsymbol{disc}_\alpha$ *and* $\dot{x} = y \ \& \ x = y$ *by using the rule* $[\cup]$*.*

$$\dfrac{\dfrac{\dfrac{x = 0 \vdash x_0 \leqslant 10 \rightarrow [\boldsymbol{disc}_\alpha]x_0 \leqslant 10 \qquad x = 0 \vdash x_0 \leqslant 10 \rightarrow [\dot{x}_0 = y \ \& \ x_0 = y]x_0 \leqslant 10}{x = 0, x_0 \leqslant 10 \vdash [\boldsymbol{disc}_\alpha \cup \dot{x}_0 = y \ \& \ x_0 = y]x_0 \leqslant 10}\ [\cup]}{x = 0 \vdash x_0 \leqslant 10 \rightarrow [\boldsymbol{disc}_\alpha \cup \dot{x}_0 = y \ \& \ x_0 = y]x_0 \leqslant 10}\ \rightarrow_r}{x = 0 \vdash \forall x, (x \leqslant 10 \rightarrow [\boldsymbol{disc}_\alpha \cup \dot{x} = y \ \& \ x = y]x \leqslant 10)}\ \forall_r$$

The sequent $x = 0 \vdash x_0 \leqslant 10 \rightarrow [\dot{x}_0 = y \ \& \ x_0 = y]x_0 \leqslant 10$ *is clearly not valid. For example, if* $x_0 = 1$ *at the initialization, then we have* $x_0 \leqslant 10$*,* $y = 1$*, and thus* $\dot{x}_0 = 1$*. As* x_0 *is evolving linearly, we have that the formula* $x_0 \leqslant 10$ *will not hold at some point.*

Proof. The behavior α is of the form $(\mathbf{disc}_\alpha)^*$ since it is a purely discrete component. We assume to have a proof tree of the sequent $\mathcal{E}, A_\alpha \vdash [(\mathbf{disc}_\alpha)^*]G_\alpha$. It is the same assumption as in the proof of Theorem 2.

$$\dfrac{\dfrac{\Pi_{init_\alpha}}{\mathcal{E}, A_\alpha \vdash G_\alpha} \qquad \dfrac{\dfrac{\dfrac{\Pi_{ind_\alpha}}{\mathcal{E}, A_\alpha, G_\alpha^\alpha \vdash [\mathbf{disc}_\alpha^\alpha]G_\alpha^\alpha}}{\mathcal{E}, A_\alpha \vdash G_\alpha^\alpha \rightarrow [\mathbf{disc}_\alpha^\alpha]G_\alpha^\alpha}\ \rightarrow_r}{\mathcal{E}, A_\alpha \vdash \forall^\alpha (G_\alpha \rightarrow [\mathbf{disc}_\alpha]G_\alpha)}\ \forall_r}{\mathcal{E}, A_\alpha \vdash [(\mathbf{disc}_\alpha)^*]G_\alpha}\ [\text{Ind}]$$

The behavior β of the continuous component B is of the form $\dot{Y} = \theta_Y \ \& \ H_Y$. We assume to have a proof tree of the sequent $\mathcal{E}, A_\beta \vdash [(\dot{Y} = \theta_Y \ \& \ H_Y)^*]G_\beta$. It has the following shape.

$$\dfrac{\dfrac{\Pi_{init_\beta}}{\mathcal{E}, A_\beta \vdash G_\beta} \quad \dfrac{\dfrac{\dfrac{\Pi_{step_\beta}}{\mathcal{E}, A_\beta, G_\beta^\beta \vdash [(\dot{Y} = \theta_Y)^\beta \ \& \ H_Y^\beta]G_\beta^\beta}}{\mathcal{E}, A_\beta \vdash G_\beta^\beta \rightarrow [(\dot{Y} = \theta_Y)^\beta \ \& \ H_Y^\beta]G_\beta^\beta} \rightarrow_r}{\mathcal{E}, A_\beta \vdash \forall^\beta(G_\beta \rightarrow [\dot{Y} = \theta_Y \ \& \ H_Y]G_\beta)} \forall_r}{\mathcal{E}, A_\beta \vdash [(\dot{Y} = \theta_Y \ \& \ H_Y)^*]G_\beta} [\text{Ind}]$$

We first apply the induction rule and then the skolemization step (rule $\forall_r$). Notice that occurrences of bound variables of β in H_Y are replaced by fresh variables, which is the meaning of the notation H_Y^β.

Recall that the parallel composition of α and β results in the behavior $\big(\mathbf{disc}_\alpha \cup (\dot{Y} = \theta_Y \ \& \ H_Y)\big)^*$. We provide a proof tree for the sequent $\mathcal{E}, A_\alpha, A_\beta \vdash [\big(\mathbf{disc}_\alpha \cup (\dot{Y} = \theta_Y \ \& \ H_Y)\big)^*](G_\alpha \wedge G_\beta)$.

We first apply the induction rule [Ind] with $(G_\alpha \wedge G_\beta)$ as invariant.

$$\dfrac{\mathcal{E}, A_\alpha, A_\beta \vdash G_\alpha \wedge G_\beta \qquad \mathcal{E}, A_\alpha, A_\beta \vdash \forall^{\alpha\beta}\big((G_\alpha \wedge G_\beta) \rightarrow [\mathbf{disc}_\alpha \cup (\dot{Y} = \theta_Y \ \& \ H_Y)](G_\alpha \wedge G_\beta)\big)}{\mathcal{E}, A_\alpha, A_\beta \vdash [\big(\mathbf{disc}_\alpha \cup (\dot{Y} = \theta_Y \ \& \ H_Y)\big)^*](G_\alpha \wedge G_\beta)}$$

We obtain two sub-goals, one corresponding for the initial step and one for the induction step. The first sub-goal is handled as in the proof of the previous Theorem 2. We split the formula $G_\alpha \wedge G_\beta$ using the rule $\wedge$ and then use the proof tree Π_{init_α} and Π_{init_β} to conclude.

We consider the second sub-goal $\mathcal{E}, A_\alpha, A_\beta \vdash \forall^{\alpha\beta}\big((G_\alpha \wedge G_\beta) \rightarrow [\mathbf{disc}_\alpha \cup (\dot{Y} = \theta_Y \ \& \ H_Y)](G_\alpha \wedge G_\beta)\big)$. We apply the rule $\forall_r$ to replace all occurrences of bound variables of α and β in the formula $(G_\alpha \wedge G_\beta) \rightarrow [\mathbf{disc}_\alpha \cup (\dot{Y} = \theta_Y \ \& \ H_Y)](G_\alpha \wedge G_\beta)$. We denote by $G_\alpha^{\alpha\beta}$ the effect of this replacement on the formula G_α.

The step (i) is the application of the separation Lemma 1. Indeed, from conditions $(b1)$ and $(b2)$, we deduce that $G_\alpha^{\alpha\beta}$ is equivalent to G_α^α. It justifies the fact that we obtain the sequent $\mathcal{E}, A_\alpha, A_\beta \vdash (G_\alpha^\alpha \wedge G_\beta^\beta) \rightarrow [\mathbf{disc}_\alpha^{\alpha\beta} \cup \big((\dot{Y} = \theta_Y)^{\alpha\beta} \ \& \ H_Y^\beta\big)](G_\alpha^\alpha \wedge G_\beta^\beta)$. We apply successively the rules $\rightarrow_r$ and $\wedge_l$ to pass the formula $(G_\alpha^\alpha \wedge G_\beta^\beta)$ into the hypothesis.

$$\dfrac{\dfrac{\dfrac{\dfrac{\mathcal{E}, A_\alpha, A_\beta, G_\alpha^\alpha, G_\beta^\beta \vdash [\mathbf{disc}_\alpha^{\alpha\beta} \cup \big((\dot{Y} = \theta_Y)^{\alpha\beta} \ \& \ H_Y^\beta\big)](G_\alpha^\alpha \wedge G_\beta^\beta)}{\mathcal{E}, A_\alpha, A_\beta, G_\alpha^\alpha \wedge G_\beta^\beta \vdash [\mathbf{disc}_\alpha^{\alpha\beta} \cup \big((\dot{Y} = \theta_Y)^{\alpha\beta} \ \& \ H_Y^\beta\big)](G_\alpha^\alpha \wedge G_\beta^\beta)} \wedge_l}{\mathcal{E}, A_\alpha, A_\beta \vdash (G_\alpha^\alpha \wedge G_\beta^\beta) \rightarrow [\mathbf{disc}_\alpha^{\alpha\beta} \cup \big((\dot{Y} = \theta_Y)^{\alpha\beta} \ \& \ H_Y^\beta\big)](G_\alpha^\alpha \wedge G_\beta^\beta)}}{\mathcal{E}, A_\alpha, A_\beta \vdash (G_\alpha^{\alpha\beta} \wedge G_\beta^{\alpha\beta}) \rightarrow [\mathbf{disc}_\alpha^{\alpha\beta} \cup \big((\dot{Y} = \theta_Y)^{\alpha\beta} \ \& \ H_Y^\beta\big)](G_\alpha^{\alpha\beta} \wedge G_\beta^{\alpha\beta})} (i)}{\mathcal{E}, A_\alpha, A_\beta \vdash \forall^{\alpha\beta}\big((G_\alpha \wedge G_\beta) \rightarrow [\mathbf{disc}_\alpha^{\alpha\beta} \cup \big((\dot{Y} = \theta_Y)^{\alpha\beta} \ \& \ H_Y^\beta\big)](G_\alpha \wedge G_\beta)\big)} \forall_r$$

Our next step is to split the hybrid program $\mathbf{disc}_\alpha^{\alpha\beta} \cup \big((\dot{Y} = \theta_Y)^{\alpha\beta} \ \& \ H_Y^\beta\big)$ in two with the non-deterministic choice rule $[\cup]$. We have two cases to consider: the execution of the discrete part and the execution of the continuous part.

$$\frac{\mathcal{E}, A_\alpha, A_\beta, G_\alpha^\alpha, G_\beta^\beta \vdash [\mathbf{disc}_\alpha^{\alpha\beta}](G_\alpha^\alpha \wedge G_\beta^\beta) \qquad \mathcal{E}, A_\alpha, A_\beta, G_\alpha^\alpha, G_\beta^\beta \vdash [(\dot{Y} = \theta_Y)^{\alpha\beta} \ \& \ H_Y^\beta](G_\alpha^\alpha \wedge G_\beta^\beta)}{\mathcal{E}, A_\alpha, A_\beta, G_\alpha^\alpha, G_\beta^\beta \vdash [\mathbf{disc}_\alpha^{\alpha\beta} \cup \big((\dot{Y} = \theta_Y)^{\alpha\beta} \ \& \ H_Y^\beta\big)](G_\alpha^\alpha \wedge G_\beta^\beta)} [\cup]$$

The left sub-goal, $\mathcal{E}, A_\alpha, A_\beta, G_\alpha^\alpha, G_\beta^\beta \vdash [\mathbf{disc}_\alpha^{\alpha\beta}](G_\alpha^\alpha \wedge G_\beta^\beta)$, is handled as in the proof of the Theorem 2. We detail the proof tree for the right sub-goal, $\mathcal{E}, A_\alpha, A_\beta, G_\alpha^\alpha, G_\beta^\beta \vdash [(\dot{Y} = \theta_Y)^{\alpha\beta} \ \& \ H_Y^\beta](G_\alpha^\alpha \wedge G_\beta^\beta)$.

We first split the formula $G_\alpha^\alpha \wedge G_\beta^\beta$ under the modality with the rule [BoxAnd]. We then carefully consider the two sub-goals.

$$\frac{\mathcal{E}, A_\alpha, A_\beta, G_\alpha^\alpha, G_\beta^\beta \vdash [(\dot{Y} = \theta_Y)^{\alpha\beta} \ \& \ H_Y^\beta]G_\alpha^\alpha \qquad \mathcal{E}, A_\alpha, A_\beta, G_\alpha^\alpha, G_\beta^\beta \vdash [(\dot{Y} = \theta_Y)^{\alpha\beta} \ \& \ H_Y^\beta]G_\beta^\beta}{\mathcal{E}, A_\alpha, A_\beta, G_\alpha^\alpha, G_\beta^\beta \vdash [(\dot{Y} = \theta_Y)^{\alpha\beta} \ \& \ H_Y^\beta](G_\alpha^\alpha \wedge G_\beta^\beta)} \text{[BoxAnd]}$$

For the first sub-goal, $\mathcal{E}, A_\alpha, A_\beta, G_\alpha^\alpha, G_\beta^\beta \vdash [(\dot{Y} = \theta_Y)^{\alpha\beta} \ \& \ H_Y^\beta]G_\alpha^\alpha$, remember that G_α does not contain any occurrences of bound variables of $\dot{Y} = \theta_Y \ \& \ H_Y$. Thus the evolution of the differential equation does not affect G_α^α. We symbolically execute it to obtain the sequent which is closed by the axiom rule.

$$\frac{\overline{\mathcal{E}, A_\alpha, A_\beta, G_\alpha^\alpha, G_\beta^\beta \vdash G_\alpha^\alpha}\ \text{ax}}{\mathcal{E}, A_\alpha, A_\beta, G_\alpha^\alpha, G_\beta^\beta \vdash [(\dot{Y} = \theta_Y)^{\alpha\beta} \ \& \ H_Y^\beta]G_\alpha^\alpha} \text{Symbolic execution}$$

It remains to prove the sequent $\mathcal{E}, A_\alpha, A_\beta, G_\alpha^\alpha, G_\beta^\beta \vdash [(\dot{Y} = \theta_Y)^{\alpha\beta} \ \& \ H_Y^\beta]G_\beta^\beta$. Using the condition $(c2)$, we obtain A_β^β in the hypothesis. We conclude with the branch Π_{step_β}.

□

3.3.6 For two general components

We have shown in Subsection 3.3.3 how to retain contracts in the case of the parallel composition of two continuous components. We have proved a similar theorem in Subsection 3.3.4 for the case of the parallel composition of two purely discrete components. In Subsection 3.3.5, we have proved how to retain contracts when we compose discrete component with continuous component. We state the main theorem which reunite the previous results for the case of the parallel composition of general components, *i.e.* with both discrete and continuous aspects.

Theorem. *Let* `A` *and* `B` *be two components with behaviors of the general form and respective contracts* (A_α, G_α) *and* (A_β, G_β)*. Assume that they both satisfy their contracts. Furthermore, assume that*

(a) `A.outputs` $\cap$ `B.outputs` $= \varnothing$,

(b_1) `A.outputs` *is separated from* G_β

(b_2) `B.outputs` *is separated from* G_α,

(c_1) G_α *must implies the assumptions* A_β *that refers to outputs of* `A`

(c_2) G_β *must implies the assumptions* A_α *that refers to outputs of* `B`.

Then the component AB *resulting from the parallel composition of* A *and* B *satisfies the conjunction of contracts* $(A_\alpha \wedge A_\beta, G_\alpha \wedge G_\beta)$.

Recall that the behavior α of a general component is of the form $(\mathbf{disc}_\alpha \cup \mathbf{cont}_\alpha)^*$, where $\mathbf{cont}_\alpha$ is a differential equation with evolution domain H_X.

Theorem 4. *Let α and β be two behaviors of general form of components* A *and* B *with respective contracts (A_α, G_α) and (A_β, G_β). Assume that we have two proof trees of $\mathcal{E}, A_\alpha \vdash [\alpha]G_\alpha$ and $\mathcal{E}, A_\beta \vdash [\beta]G_\beta$ respectively, where $\mathcal{E}$ is the environment. Furthermore, assume that*

(*a*) $BV(\alpha) \cap BV(\beta) = \varnothing$,

(b_1) $BV(\alpha) \cap FV(G_\beta) = \varnothing$ *and* $BV(\alpha) \cap FV(H_Y) = \varnothing$,

(b_2) $BV(\beta) \cap FV(G_\alpha) = \varnothing$ *and* $BV(\beta) \cap FV(H_X) = \varnothing$,

(c_1) $A_\alpha \vdash \forall^\beta(G_\beta \rightarrow A_\alpha)$,

(c_2) $A_\beta \vdash \forall^\alpha(G_\alpha \rightarrow A_\beta)$.

Then it exists a proof tree of $\mathcal{E}, A_\alpha, A_\beta \vdash [\alpha \circ \beta](G_\alpha \wedge G_\beta)$.

The proof uses the same mechanism as in the proofs of Theorems 1, 2 and 3.

Proof. Recall that the behavior α of component A is of the form $(\mathbf{disc}_\alpha \cup \mathbf{cont}_\alpha)^*$. As in previous theorems, we assume to have a proof tree of the sequent $\mathcal{E}, A_\alpha \vdash [(\mathbf{disc}_\alpha \cup \mathbf{cont}_\alpha)^*]G_\alpha$. It is of the following form:

$$\dfrac{\dfrac{\Pi_{init_\alpha}}{\mathcal{E}, A_\alpha \vdash G_\alpha} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\Pi_{ind_{\mathbf{disc}_\alpha}}}{\mathcal{E}, A_\alpha, G_\alpha^\alpha \vdash [\mathbf{disc}_\alpha^\alpha]G_\alpha^\alpha} \quad \dfrac{\Pi_{ind_{\mathbf{cont}_\alpha}}}{\mathcal{E}, A_\alpha, G_\alpha^\alpha \vdash [\mathbf{cont}_\alpha^\alpha]G_\alpha^\alpha}}{\mathcal{E}, A_\alpha, G_\alpha^\alpha \vdash [\mathbf{disc}_\alpha^\alpha \cup \mathbf{cont}_\alpha^\alpha]G_\alpha^\alpha}[\cup]}{\mathcal{E}, A_\alpha \vdash G_\alpha^\alpha \rightarrow [\mathbf{disc}_\alpha^\alpha \cup \mathbf{cont}_\alpha^\alpha]G_\alpha^\alpha}\rightarrow_r}{\mathcal{E}, A_\alpha \vdash \forall^\alpha(G_\alpha \rightarrow [\mathbf{disc}_\alpha \cup \mathbf{cont}_\alpha]G_\alpha}\forall_r}{\mathcal{E}, A_\alpha \vdash [(\mathbf{disc}_\alpha \cup \mathbf{cont}_\alpha)^*]G_\alpha}[\text{Ind}]$$

We have the same for the proof of the sequent $\mathcal{E}, A_\beta \vdash [(\mathbf{disc}_\beta \cup \mathbf{cont}_\beta)^*]G_\beta$ Recall that the behavior of the resulting component is $\alpha \circ \beta \triangleq \big(\mathbf{disc}_\alpha \cup \mathbf{disc}_\beta \cup (\mathbf{cont}_\alpha \circ_c \mathbf{cont}_\beta)\big)^*$.

We apply the induction rule [Ind]. We obtain two sub-goals: the initial step (i) and the induction step (ii). The first step (i) is handled exactly as for the discrete case or for the continuous case with rule [DI].

For the last sequent $\mathcal{E}, A_\alpha, A_\beta \vdash \forall^{\alpha\beta}\big((G_\alpha \wedge G_\beta) \rightarrow [(\mathbf{disc}_\alpha \cup \mathbf{disc}_\beta) \cup \mathbf{cont}_{\alpha\beta}](G_\alpha \wedge G_\beta)\big)$, we first apply the skolemization rule, and then the non-deterministic choice rule $[\cup]$ (cf Figure 3.17).

As shown in the Figure 3.17, we have two premises to consider. The left premise $\mathcal{E}, A_\alpha, A_\beta, G_\alpha^{\alpha\beta} \wedge G_\beta^{\alpha\beta} \vdash [\mathbf{disc}_\alpha^{\alpha\beta} \cup \mathbf{disc}_\beta^{\alpha\beta}](G_\alpha^{\alpha\beta} \wedge G_\beta^{\alpha\beta})$ is exactly similar to the induction step of the case where both components are discrete and is closed using $\Pi_{ind_{\mathbf{disc}_\alpha}}$ and $\Pi_{ind_{\mathbf{disc}_\beta}}$. The right premise $\mathcal{E}, A_\alpha, A_\beta, (G_\alpha^{\alpha\beta} \wedge G_\beta^{\alpha\beta}) \vdash [\mathbf{cont}_{\alpha\beta}^{\alpha\beta}](G_\alpha^{\alpha\beta} \wedge G_\beta^{\alpha\beta})$ is closed by applying the Theorem 1 and using the proof trees $\Pi_{ind_{\mathbf{cont}_\alpha}}$ and $\Pi_{ind_{\mathbf{cont}_\beta}}$.

□

$$\dfrac{(ii)}{\underbrace{\mathcal{E}, A_\alpha, A_\beta \vdash \forall^{\alpha\beta}\big((G_\alpha \wedge G_\beta) \to [(\mathbf{disc}_\alpha \cup \mathbf{disc}_\beta) \cup (\mathbf{cont}_\alpha \circ_c \mathbf{cont}_\beta)](G_\alpha \wedge G_\beta)\big)}}$$

$$\dfrac{(i)}{\underbrace{\mathcal{E}, A_\alpha, A_\beta \vdash G_\alpha \wedge G_\beta}}$$

$$\dfrac{}{\mathcal{E}, A_\alpha, A_\beta \vdash [(\mathbf{disc}_\alpha \cup \mathbf{disc}_\beta \cup \mathbf{cont}_\alpha \circ_c \mathbf{cont}_\beta)^*](G_\alpha \wedge G_\beta)}\ [\text{Ind}]$$

Figure 3.16: Application of the rule [Ind] for the general case

$$\underbrace{\mathcal{E}, A_\alpha, A_\beta, G_\alpha^{\alpha\beta} \wedge G_\beta^{\alpha\beta} \vdash [\mathbf{cont}_{\alpha\beta}^{\alpha\beta}](G_\alpha^{\alpha\beta} \wedge G_\beta^{\alpha\beta})}$$

$$\underbrace{\mathcal{E}, A_\alpha, A_\beta, G_\alpha^{\alpha\beta} \wedge G_\beta^{\alpha\beta} \vdash [\mathbf{disc}_\alpha^{\alpha\beta} \cup \mathbf{disc}_\beta^{\alpha\beta}](G_\alpha^{\alpha\beta} \wedge G_\beta^{\alpha\beta})}$$

$$\dfrac{\dfrac{\dfrac{}{\mathcal{E}, A_\alpha, A_\beta, G_\alpha^{\alpha\beta} \wedge G_\beta^{\alpha\beta} \vdash [(\mathbf{disc}_\alpha^{\alpha\beta} \cup \mathbf{disc}_\beta^{\alpha\beta}) \cup \mathbf{cont}_{\alpha\beta}^{\alpha\beta}](G_\alpha^{\alpha\beta} \wedge G_\beta^{\alpha\beta})}\ [\cup]}{\mathcal{E}, A_\alpha, A_\beta \vdash (G_\alpha^{\alpha\beta} \wedge G_\beta^{\alpha\beta}) \to [(\mathbf{disc}_\alpha^{\alpha\beta} \cup \mathbf{disc}_\beta^{\alpha\beta}) \cup \mathbf{cont}_{\alpha\beta}^{\alpha\beta}](G_\alpha^{\alpha\beta} \wedge G_\beta^{\alpha\beta})}\ \to_r}{\mathcal{E}, A_\alpha, A_\beta \vdash \forall^{\alpha\beta}\big((G_\alpha \wedge G_\beta) \to [(\mathbf{disc}_\alpha \cup \mathbf{disc}_\beta) \cup \mathbf{cont}_{\alpha\beta}](G_\alpha \wedge G_\beta)\big)}\ \forall_r$$

Figure 3.17: Induction step

The proof follows the distinction between the discrete part and the continuous one. We now consider the proof of the `Cruise-Control` component.

Example 33 (Contract of the `cruise-control`)**.** *The* `Tachymeter` *and* `Engine` *components respect the conditions of the Theorem 4:*

- *The outputs are separated. We have* $BV(\boldsymbol{Engine}) = \{s_{eng}\}$ *and* $BV(\boldsymbol{Tach}) = \{s_{acm}, s_{tach}, a\}$*, thus* $BV(\boldsymbol{Engine}) \cap BV(\boldsymbol{Tach}) = \varnothing$*.*
- *The guarantees do not refers to exterior outputs. We have* $Var(G_{eng}) = \{s_{eng}\}$ *and* $Var(G_{tach}) = \{s_{acm}, s_{tach}, a\}$*. It is easy to check that* $BV(\boldsymbol{Engine}) \cap Var(G_{tach}) = \varnothing$ *and* $BV(\boldsymbol{Tach}) \cap Var(G_{eng}) = \varnothing$*.*
- *The guarantees imply the assumptions. We have* $A_{eng} \triangleq G_{tach}$ *and* $A_{tach} \triangleq G_{eng}$*. It is thus trivial to verify* $A_{eng} \vdash \forall^{tach}(G_{tach} \to A_{eng})$ *and* $A_{tach} \vdash \forall^{eng}(G_{tach} \to A_{eng})$*.*

Thus, by application of the Theorem 4, the following sequent is valid:

$$\mathcal{E}, A_{cruise} \vdash [\boldsymbol{Cruise\text{-}control}]G_{cruise}$$

where $A_{cruise} \triangleq A_{eng} \wedge A_{tach}$ and $G_{cruise} \triangleq G_{eng} \wedge G_{tach}$. The `Cruise-Control` *satisfies its contract, and we have obtained this proof automatically.*

We have exhibited in this section a methodology to lift the contracts of individual components through composition. If components `A` and `B` satisfies their respective contracts (A_α, G_α) and (A_β, G_β), then the compound component `AB` resulting from the parallel composition satisfies the contract $(A_\alpha \wedge A_\beta, G_\alpha \wedge G_\beta)$.

The next section is devoted to the presentation of the implementation of a prototype in KeYmaera X. We have successfully passed the cruise-controller example.

3.4 A prototype in KeYmaera X

We have implemented a prototype using the tactic language Bellerophon in KeYmaera X to demonstrate the feasibility of our approach. We present the tactic language Bellerophon in Subsection 3.4.1 and exemplify with the Fibonacci Example 1. We present the structure of our prototype in Subsection 3.4.2. The cruise-controller have been verified using this example.

3.4.1 Presentation of Bellerophon

Bellerophon is a tactic language for the proof of hybrid systems and a library implemented in KeYmaera X. It is both a stand-alone language and a Domain-Specific Language in the Scala programming language [46, 47].

It has been developed to provide decision procedures and heuristics for the theorem proving of hybrid systems in KeYmaera X. It aims to ease proof automation.

It uses built-in tactics that can be combined through several tactic combinators. Built-in tactics directly manipulate the KeYmaera X core to transform formulas in a validity-preserving manner. They correspond either to specific rules of the sequent calculus, *e.g.* the tactic `andR` which corresponds to the rule $\wedge_r$, or correspond to heuristics, *e.g.* the tactic `prop` corresponds to the repeated application of propositional rules until they are not applicable.

Definition 28 (Tactic combinators)**.**

$$e_1, e_2 ::= \tau \mid e(\overline{v}) \mid e_1 \& e_2 \mid e_1 | e_2 \mid e^* \mid ?(e) \mid < (e_1, e_2, \ldots, e_n) \mid \texttt{abbrv} P(\overline{x}) = \varphi \texttt{ in } e$$

τ is a built-in tactic. $e(\overline{v})$ applies a tactic e to a list of positions or formulas. $e_1 \& e_2$ is the sequential composition of two tactics; it applies the tactic e_2 on the output of e_1. $e_1 | e_2$ applies e_2 if e_1 fails. e^* repeatedly applies the tactic e as long as it is applicable. $?(e)$ applies e if it does not result in a error. $< (e_1, e_2, \ldots, e_n)$ is used when there is n subgoals; it applies e_1 to the first subgoal, e_2 to the second subgoal, etc. $\texttt{abbrv} P(\overline{x}) = \varphi \texttt{ in } e$ replaces all occurrences of φ with $P(\overline{x})$ in the current subgoal, and then applies e. After e, remaining occurrences of $P(\overline{x})$ are uniformly substituted back to φ.

Example 34 (Fibonacci example)**.** *Remember that the hybrid program representing the Fibonacci sequence is Fibonacci $\triangleq (F_n := F_{n+1}; F_{n+1} := F_{n+2}; F_{n+2} := F_{n+1} + F_n)^*$ (cf Definition 1) and we want to prove the following formula $(F_n = 0 \wedge F_{n+1} = 1 \wedge F_{n+2} = 1) \rightarrow [Fibonacci] F_{n+2} = F_{n+1} + F_n$ (cf proof tree 2.33).*

We can find the .kyx file of the Fibonacci example in the annex 5.3.2. It is proved in KeYmaera X by the following tactic:

```
implyR(1) ; loop({`Fn2=Fn1+Fn`}, 1) ; <(
 QE,
 closeId,
 composeb(1) ; assignb(1) ; composeb(1) ; assignb(1) ; assignb(1) ; QE
)
```

The tactic `implyR(1)` *is the application of the rule* $\rightarrow_r$ *to pass the initial values in the hypothesis. Then, the tactic* `loop({`Fn2=Fn1+Fn`}, 1)` *apply the rule [Ind2] with* $Fn2 = Fn1+Fn$ *as invariant. It opens three sub-goals.*

The first one requires that the invariant $Fn2 = Fn1 + Fn$ *holds at the initialization. We apply the rule QE to apply Real Arithmetic and conclude. The second sub-goal is trivial since the invariant introduced in [Ind2] is the same as the original formula.*

The third sub-goal deal with the induction step. We deconstruct step-by-step the hybrid program by applying the tactics `composeb(1)` *and* `assignb(1)` *corresponding to rules* $[;]$ *and* $[:=]$. *We conclude then by the rule QE.*

3.4.2 Implementation in KeYmaera X

We detail how we have implemented the process described in the proof of Theorems 1, 2, 3 and 4. We define a tactic corresponding for each case; one for the case where both components are continuous, one where both are discrete, one for the case of one discrete component and one continuous component and one for the case of general components.

Continuous-continuous case

We follow the structure of the proof of the Theorem 1. In the theorem, we assume to have a proof of the satisfaction of contract for each component. We have discussed that the proof can be achieved by using either the Differential Induction rule [DI], the rule [ODESolve] where we provide a solution of the differential equation and the Differential Weakening rule [DW]. We consider each association that can arise, *e.g.* the use of rule [DI] for both proof of satisfaction of their respective contracts by components, and show how to exhibit a proof tree from them.

We define a tactic `continuousComposition` which try a sub-tactic for each possible association. We have a sub-tactic for the case where both proofs use the Differential Induction rule [DI], one for the case where both proofs use the Differential Weakening rule [DW], and one for the case where one proof use the Differential Induction rule and the other one the Differential Weakening rule. We do not consider the case where we use the rule [ODESolve] since we can always relate it to the case of Differential Induction rule [DI]. Finally, we combine each sub-tactic with the combinator |.

```
val continuousComposition: BelleExpr = (
     di2 | dw2 | dwdi | didw
     )
```

We detail each sub-tactic `di2`, `dw2`, `dwdi` and `didw`.

DI-DI We consider the case where both cases have been proved with the differential induction rule [DI]. We require the user to provide the tactic used to proved the initial step (tactic `baseStep1`) and the differential induction step (tactic `diffInductionStep1`). They correspond to the proof trees P^{H_X} and P'_α in the proof of Theorem 1. We have the same for the second component (`baseStep2` and `diffInductionStep2`).

```
/* Base case of the first tactic */
   val baseStep1: BelleExpr = (
     /* To fill by the user */
     ...
   )

/* Differential induction step of the first tactic */
   val diffInductionStep1: BelleExpr = (
     /* To fill by the user */
     ...
   )
```

We follow the procedure presented in the proof of the Theorem 1. We apply the Differential Induction rule [DI] (tactic `dI('diffInd)(1)`) and consider the initial step and the differential induction step. We split the goal corresponding to the initial step with the rule $\wedge_r$ (tactic `andR('R)`) and use our two assumptions `baseStep1` and `baseStep2` to conclude. In the induction step, the sequent is of the form $\mathcal{E}, A_\alpha, A_\beta \vdash [\dot{X} := \theta_X; \dot{Y} := \theta_Y]G'_\alpha \wedge G'_\beta$. We perform all the assignments with the tactic `chase('R)` and split the resulting goal in two. We use then our assumptions `diffInductionStep1` and `diffInductionStep2`.

```
val di2: BelleExpr = (
      print("Try DI2 tactic") & implyR(1) & dI('diffInd)(1) <(
        print("Base step") & andR('R) <(
          baseStep1,
          baseStep2
        ),
        print("Differential Induction step") & chase('R) & andR('R) <(
          diffInductionStep1,
          diffInductionStep2
        )
      )
    )
```

The Differential Induction rule in KeYmaera X is implemented by the tactic `dI` with several degrees of automation. We have chosen to consider only the case corresponding to the version of the rule we have presented in the Chapter 2, the other cases are for future works.

DW-DW We consider the case where the proof of satisfaction of contracts of both components have been achieved with the Differential Weakening rule [DW]. We follow the structure of the proof of the Theorem 1. First, we require the user to provide the tactics `tacticDW1` and `tacticDW2` used to achieve each respective proof.

```
val tacticDW1: BelleExpr = (
     /* To fill by the user */
     ...
     )

val tacticDW2: BelleExpr = (
     /* To fill by the user */
     ...
     )
```

We apply the tactic `dW('R)` corresponding to the Differential Weakening rule [DW]. We use the tactics `tacticDW1` and `tacticDW2` to close the proof.

```
val dw2: BelleExpr = (
     print("Try DW2 tactic") & implyR(1) & dW('R)
       & implyR(1) & andR(1) <(
         tacticDW1,
         tacticDW2
       )
     )
```

DI-DW We consider the case where the proof of satisfaction of contract of the first component has been achieved with the Differential Induction rule [DI] and the one for the second component has been achieved with the Differential Weakening rule [DW]. We require the user to provide the tactic used to proved the initial step (tactic `baseStep1`) and the differential induction step (tactic `diffInductionStep1`). We require also that she provides the guarantee `invariant1` of the first contract and the tactic `tacticDW2` used to achieve the second proof.

```
/* Base case of the first tactic */
   val baseStep1: BelleExpr = (
     /* To fill by the user */
     ...
   )

/* Differential induction step of the first tactic */
   val diffInductionStep1: BelleExpr = (
     /* To fill by the user */
     ...
   )
```

```
val invariant1: Formula =
   /* To fill by the user */
     ...

val tacticDW2: BelleExpr = (
     /* To fill by the user */
     ...
     )
```

We follow the structure of the proof of the Theorem 1. We first add the guarantee of the first component into the evolution domain using the Differential Cut rule [DC] (tactic `dC(invariant1)(1)`). It results in two sub-goals. We close the first sub-goal by applying the Differential Weakening rule [DW] (tactic `dW('R)`) and split the current goal in two parts. The first part is trivial to handle since it requires to prove the formula G_α (equivalent to `invariant1`) under the assumption G_α, hence the use of the tactic `master()`. The second part is closed by using our assumptions `baseStep1` and `diffInductionStep1`.

```
val didw: BelleExpr = (
     print("Try DIDW tactic") & implyR(1) & dC(invariant1)(1) <(
       dW('R) & implyR('R) & andR(1) <(
         master(),
         tacticDW2
       ),
       dI('diffInd)(1) <(
       print("Base step") & baseStep1,
       print("Differential Induction step") & chase('R) & diffInductionStep1,
       )
     )
    )
```

We define also a symmetrical tactic `dwdi` for the inverse case.

Conclusion We have presented how to implement several of the possible associations considered in the proof of the Theorem 1. The implementation is not complete and does not cover every possibility in KeYmaera X, but it shows the feasibility of such implementation.

Discrete-discrete case

We present the tactic corresponding to the case where both components are purely discrete. The behavior α of the first component is assumed to be purely discrete and is of the form $\alpha \triangleq (\mathbf{disc}_\alpha)^*$. Given the contract (A_α, G_α), its satisfaction is obtained by the proof of the sequent $\mathcal{E}, A_\alpha \vdash [(\mathbf{disc}_\alpha)^*]G_\alpha$. The proof tree is obtained by the use of Induction rule [Ind] and is closed by the branches Π_{Init_α} and Π_{Step_α}. We require to the user to provide the tactics corresponding to these two steps. We also require to have the guarantee G_α.

```
/* Base case of the first tactic */
   val baseStep1: BelleExpr = (
     /* To fill by the user */
     ...
   )

/* Induction step of the second tactic */
   val inductionStep1: BelleExpr = (
     /* To fill by the user */
     ...
     )

val invariant1: Formula =
     /* To fill by the user */
     ...
```

We have similar tactics (`baseStep2`, `inductionStep2` and `invariant2`) for the other component. The tactic to obtain the proof of satisfaction of the conjunction of contracts by the composition of components is given in the tactic `discDiscCase`.

```
val discDiscCase: BelleExpr = (
     print("Try Discrete-Discrete tactic") & implyR(1)
     & loop(And(invariant1, invariant2))('R) <(
       print("Base step") & andR('R) <(
         baseStep1,
         baseStep2
       ),
       print("Use step") & master(),
       print("Induction step") & inductionStepTactic
     )
   )
```

In a .kyx file, the sequent is of the form $\vdash A \rightarrow [\alpha]G$. We apply the rule $\rightarrow_r$ (tactic `implyR(1)` to pass the left-hand side of the implication on the hypothesis. We then apply the induction rule [Ind] with the conjunction of guarantees. It yields three steps: the initial step, the use step and the induction step. For the first one, we split the goal in two with the rule $\wedge_r$ (tactic `andR('R)`) and then use our assumptions (tactics `baseStep1` and `baseStep2`). The second one corresponds to the case where we want to strengthen the invariant, but we do not need it here, hence the use of the tactic `master()` to close this trivial goal. For the third one, we apply the induction tactic detailed below.

```
val inductionStepTactic: BelleExpr = (
     choiceb(1) & andR('R) <(
         boxAnd(1) & andR('R) <(
           inductionStep1,
           V(1) & master()
         ),
         boxAnd(1) & andR('R) <(
           V(1) & master(),
           inductionStep2
         )
       )
   )
```

We follow the structure in the proof of Theorem 2. Recall that the sequent we want to prove is of the form $\mathcal{E}, A_\alpha, A_\beta, G_\alpha, G_\beta \vdash [\mathbf{disc}_\alpha \cup \mathbf{disc}_\beta](G_\alpha \wedge G_\beta)$. We first apply the non-deterministic choice rule $[\cup]$ (tactic `choiceb(1)`) and split the resulting goal in two with the rule $\wedge_r$ (tactic `andR('R)`). We obtain two branches which are similar. In the first one, we split the formula under the modality with the $[BoxAnd]$ rule (tactic `boxAnd(1)`) and the rule $\wedge_r$. It yields two cases; one where we have to prove that G_α holds after one execution of $\mathbf{disc}_\alpha$ and one where we have to prove that G_β holds after one execution of $\mathbf{disc}_\alpha$. We use our assumption `inductionStep1` for the first case. For the second, we apply the symbolic execution (tactic `V(1)`) and then apply the tactic `master()`.

Discrete-continuous case

The exact same tactic can be applied for the composition of a discrete component with a continuous component. Indeed, given a continuous component with behavior β and contract (A_β, G_β), we require to have a proof of the sequent $\mathcal{E}, A_\beta \vdash [\beta^*]G_\beta$ instead of a proof of the sequent $\mathcal{E}, A_\beta \vdash [\beta]G_\beta$. The tactic `inductionStep1` corresponds to the branch Π_{ind_β}. The tactic related to the continuous nature are present in the induction step.

General case

We present the tactic corresponding to the case where both components are general, *i.e.* with discrete and continuous evolution. We follow the structure of the proof of the Theorem 4. As previously, we require the user to provide the tactics `baseStep1`, `discInductionStep1` and `contInductionStep1` which corresponds to the proof trees Π_{init_α}, $\Pi_{ind_{\mathbf{disc}_\alpha}}$ and $\Pi_{ind_{\mathbf{cont}_\alpha}}$. We have the corresponding tactics `baseStep2`, `discInductionStep2` and `contInductionStep2` for the second component.

```
/* Base case of the first tactic */
val baseStep1: BelleExpr = (
  /* To fill by the user */
  ...
)
```

```
/* Discrete induction step of the first tactic */
val discInductionStep1: BelleExpr = (
  /* To fill by the user */
  ...
)

/* Continuous induction step of the first tactic */
val contInductionStep1: BelleExpr = (
  /* To fill by the user */
  ...
)
```

We also require the user to provide the guarantees `invariant1` and `invariant2` for each contract. We apply the induction rule (tactic `loop`) with the conjunction of guarantees (`And(invariant1, invariant2)`) as invariant. We consider the three resulting sub-goals. The first two are handled exactly as in the case of composition of discrete components. For the third sub-goal corresponding to the induction step, we use the non-deterministic choice rule $[\cup]$ (tactic `choiceb`) to separate the discrete part from the continuous part. We apply the tactic `inductionStepTactic` for the discrete part. It is exactly the same as in the case of purely discrete components. For the continuous part, we use the tactic `continuousComposition`.

```
val generalCase: BelleExpr = (
     print("Try General tactic") & implyR(1) & loop(And(invariant1, invariant2))('R) <(
       print("Base step") & andR('R) <(
         baseStep1,
         baseStep2
       ),
       print("Use step") & master(),
       print("Induction step") & choiceb(1) & andR('R) <(
       inductionStepTactic,
       continuousComposition
       )
     )
   )
```

Conclusion

We have presented the implementation of a prototype for our parallel composition operator. It follows the structure of the proofs of Theorems 1, 2, 3 and 4 and shows the feasibility of an implementation. Yet, it not a complete and usable implementation. We have to take care of all variations of rules in KeYmaera X. There is also other problems due to the implementation of a theoretical process into an actual implementation, *e.g.* there is a difference between formulas $G_\alpha \wedge G_\beta \wedge G_\gamma$ and $(G_\alpha \wedge G_\beta) \wedge G_\gamma$ although they are theoretically equivalent. We leave to future works such complete implementation.

3.5 Study of a water-plant example

We present a water-plant example. It is a use-case with four components to exemplify associativity. It is composed of two water-tanks with their respective controllers.

The water-tank example in Figure 3.18 is a simple example of a cyber-physical system. The water-level wl_1 inside the tank can go down or up depending if the inlet valve fin is open or not. There is a hole at the bottom of the tank which provokes an assumed constant outlet flow $fout_1$. The inlet valve is commanded by a monitor, which is a standard computer program. It decides the opening according to the value of the measured water-level wlm_1 by the sensor which repeatedly senses the value of the water-level. The controller is made of the monitor, the sensor and the actuator. It is a *time-triggered* example.

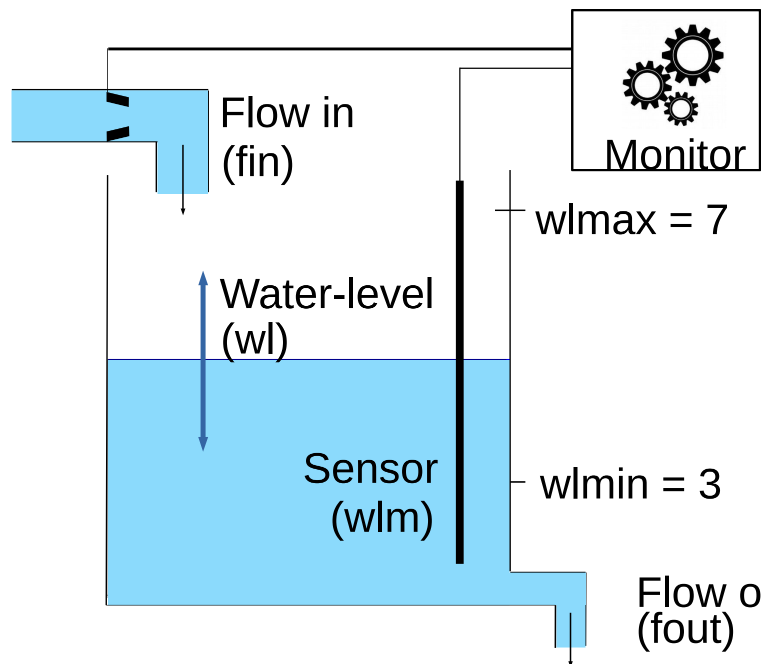


Figure 3.18: Schema of water-Tank

A second version is possible where the actuation is on the outlet flow $fout_1$ and the inlet flow fin is assumed to be constant. When connecting theses two version together, we obtain a simple water-plant as in Figure 3.19.

We detail in Subsection 3.5.1 the environment under which the water-plant operates and the initial conditions. The Subsections 3.5.2 and 3.5.3 are devoted respectively to the presentation of the water-level and of the controller component. We compose them in Subsection 3.5.4 to obtain the water-tank example. In Subsection 3.5.5, we present the water-plant obtained by the composition of two water-tanks. The last Subsection 3.5.6 discuss some limitations and the solutions that we provide in the Chapter 4.

3.5.1 Environment and initial conditions

We present the environment under which the water-plant should operate. It refers to global variables that are not modified by any components, *e.g.* the maximum water-level allowed.

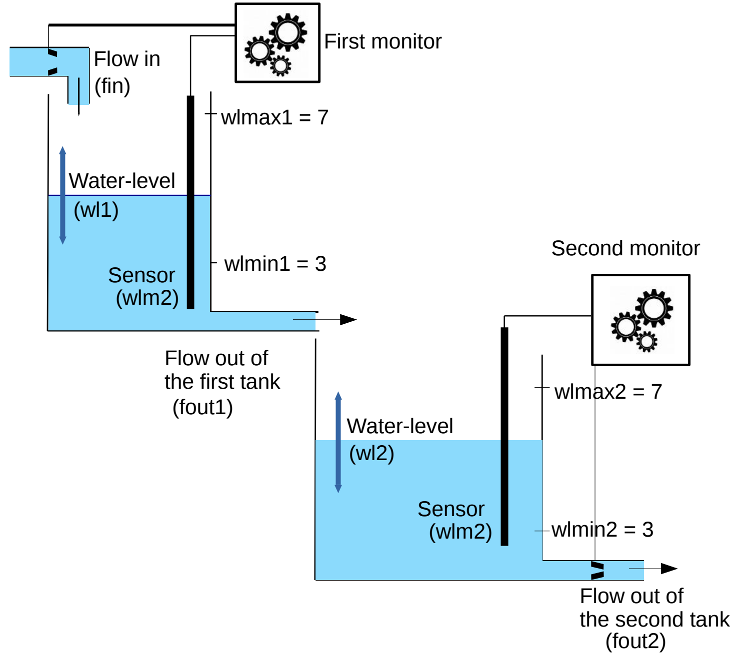


Figure 3.19: Water-plant

We precise also how the time evolves and the initial conditions of the system. We denote with $\mathcal{E}_{wp}$ the conjunction of all formulas.

Environment The *environment* is composed of parameters that are not outputs of components. The parameter δ_1 is the execution period of the first controller and δ_2 the execution period of the second controller. $wlmax1$ and $wlmin1$ are respectively the maximum threshold and the minimum threshold of the first tank that the water-level is allowed to reach. We have the same values for the second tank The outlet flow $fout_1$ of the first tank is assumed to be constant and is at 0.75.

$$\begin{cases} \delta_1 = 0.03 \\ \delta_2 = 0.02 \\ wlmax_1 = 7 \wedge wlmin_1 = 3 \\ wlmax_2 = 7 \wedge wlmin_2 = 3 \\ fout_1 = 0.75 \end{cases}$$

We will frequently replace $wlmax_1$ and $wlmin_1$ by their respective values 7 and 3.

Time We assume that the time is passing linearly and is represented by the variable t. The variable t_{ctrl_1} represents the last instant of execution of the controller. We use a similar mechanism in the cruise-control system (cf Example 26). They are both initialized to 0.

Initial conditions We provide initial values for each outputs. We assume that both water-levels have the value 5 and that the variable memorizing respective water-levels is equal to 5 also at the initialization. We also assume that the inlet valve fin and the outlet valve $fout_2$ are both closed.

$$\begin{cases} wl_1 = 5 \\ wl_2 = 5 \\ wlm_1 = 5 \\ wlm_2 = 5 \\ fin = 0 \\ fout_2 = 0 \end{cases}$$

The formula $\mathcal{E}_{wp}$ is thus the following conjunction:

$$\delta_1 = 0.03 \wedge \delta_2 = 0.02 \wedge wlmax_1 = wlmax_2 = 7 \wedge wlmin_1 = wlmin_2 = 3 \wedge fout_1 = 0.75$$
$$\wedge t = t_{ctrl_1} = t_{ctrl_2} = 0 \wedge wl_1 = wl_2 = 5 \wedge wlm_1 = wlm_2 = 5 \wedge fin = fout_2 = 0$$

3.5.2 Water-level

The `Water-level1` component is the continuous evolution of the water-level of the first tank. It depends of the inlet and outlet flow.

I/O and contract The `Water-level1` component is defined in the Figure 3.20. The inputs are the value of the measured water-level (wlm_1) by the controller and the inlet flow (fin). The assumptions A_{wl_1} correspond to the guarantees G_{ctrl_1} provided by the controller. There is only one output: the water-level wl_1. The guarantee G_{wl_1} ensures that the water-level stays within the defined range $[3, 7]$. We add the solution of the differential equation as a guarantee. It is necessary to specify the relation between the measured water-level and the real water-level because we are in a time-triggered version. It requires a deep knowledge of the timing behavior of the system. The behavior is denoted by **Wl1** and is detailed in the Definition 29.

Name: Water-level1
Inputs:
fin // inlet flow
wlm_1 // measured water-level
Assumptions:
G_{ctrl_1}
Outputs:
wl_1 // water-level
Guarantees:
$3 \leqslant wl_1 \leqslant 7$
$wl_1 = (fin - fout_1)(t - t_{ctrl_1}) + wlm_1$
Behavior:
Wl1

Figure 3.20: `Water-level` component

We can notice that the solution of the differential equation introduced as a guarantee of the `Water-level1` component refers to fin and wlm_1 which are outputs of the controller (cf Figure 3.18). It is not an issue with respect to the Theorem 4 since we show that the solution is also a guarantee of the controller.

Behavior of `Water-level1` The *behavior* of the `Water-level1` component is a differential equation. It is defined as the difference between the inlet flow fin and the outlet flow $fout_1$.

Definition 29 (Water-level behavior)**.** *The* behavior ***Wl1*** *of the water-level of the first tank is defined by the following differential equation:*

$$\dot{wl_1} = fin - fout_1, \dot{t} = 1 \ \ \& \ \ (wl_1 \geqslant 0 \wedge t \geqslant 0 \wedge 0 \leqslant t - t_{ctrl_1} \leqslant \delta_1)$$

The evolution of the component is the difference between the inlet flow fin and the outlet flow $fout_1$. We add in the evolution domain the formula $0 \leqslant t - t_{ctrl_1} \leqslant \delta_1$ which restricts the evolution of the water-level to a duration of δ_1 seconds. We use a similar mechanism in the cruise-controller example (cf Example 26).

Implementation We have implemented the component as a model in KeYmaera X. The corresponding .kyx file is in the annex 5.3.2. The proof that the component `Water-level1` satisfies its contract is achieved with the proof tactic `master` in KeYmaera X. The Figure 3.21 sum it up with the green tick meaning that we have a proof of the satisfaction of the contract.

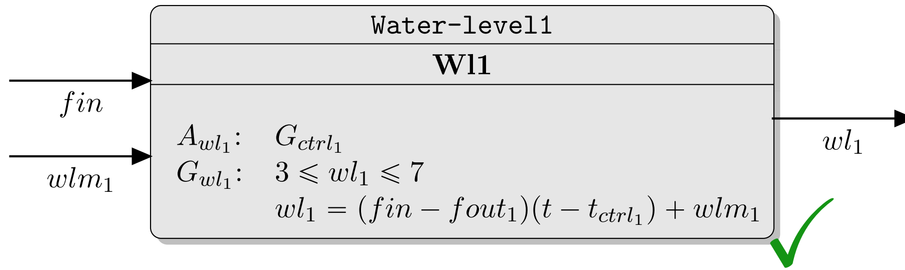


Figure 3.21: Graphical representation of the `Water-level1` component

3.5.3 Controller

The `Controller1` component regulates the water-level of the first tank. It repeatedly measures the value of the water-level at most every δ_1 units of time. It decides accordingly to this measured value to close or open the inlet valve. If the inlet valve is open, the water-level rises, otherwise it goes down.

I/O and contract The component is detailed in the Figure 3.22. The only input is the water-level wl_1. The assumption A_{ctrl_1} correspond to the guarantee of the water-level. It requires that the water-level stays in the predefined range $[3, 7]$ and that the solution of the differential equation is respected. The outputs are the measured water-level (wlm_1) and the inlet flow (fin). The guarantees G_{ctrl_1} state that wlm_1 is within the range $[3, 7]$. They also state that if the measured water-level is below 3.5 ($wlm_1 \leqslant 3.5$), then the inlet valve is opened ($fin = 1$). Conversely, if the measured water-level is above 6.5 ($wlm_1 \geqslant 3.5$), then the inlet valve is closed ($fin = 0$). If the water-level is between these two values ($3.5 \leqslant wlm_1 \leqslant 6.5$), the inlet valve can be either closed or opened ($fin = 1 \vee fin = 0$). Finally, we add the solution of the differential equation as a guarantee. It is indeed a guarantee since after the execution of the controller, we have $t = t_{ctrl_1}$ (thus $t - t_{ctrl_1} = 0$) and $wl_1 = wlm_1$.

```
Name: Controller
Inputs:
 wl_1 // measured water-level
Assumptions:
 G_{wl_1}
Outputs:
 fin // flow in
 wlm_1 // measured water-level
Guarantees:
 3 ⩽ wlm_1 ⩽ 7
 3.5 ⩾ wlm_1 → fin = 1
 wlm_1 ⩾ 6.5 → fin = 0
 (3.5 ⩽ wlm_1 ⩽ 6.5) → (fin = 0 ∨ fin = 1)
 wl_1 = (fin − fout_1)(t − t_{ctrl_1}) + wlm_1
Behavior:
 WlCtrl1
```

Figure 3.22: `Controller` component

Behavior of `Controller` The behavior **WlCtrl1** of the `Controller1` component is defined in the Definition 30.

Definition 30 (Controller behavior)**.** *The behavior **WlCtrl1** of the water-level controller of the first tank is defined by the following hybrid program:*

$$\left(\ ?t \leqslant t_{ctrl_1} + \delta_1; wlm_1 := wl_1; (?wlm_1 \geqslant 6.5; fin := 0) \cup (?wlm_1 \leqslant 3.5; fin := 1); t_{ctrl_1} := t \ \right)^*$$

It is made up of three parts : a temporal part, a sensing and the decision part. The temporal part is the test $?t \leqslant t_{ctrl_1} + \delta_1$ and the assignment $t_{ctrl_1} := t$. The test ensures that the component executes at most every δ_1 seconds. The assignment remembers the last execution. The sensing is implemented by $wlm_1 := wl_1$ where the value of the water-level wl_1 is assigned to the variable wlm_1. The decision part is a standard `if-then-else` encoding.

Implementation We have implemented the component as a model in KeYmaera X. The corresponding .kyx file is in the annex 5.3.2. The proof that the component `Controller1` satisfies its contract is achieved with the proof tactic `master` in KeYmaera X. The Figure 3.23 sum it up with the green tick meaning that we have a proof of the satisfaction of the contract.

3.5.4 Water-tank

The `Water-tank1` component is obtained by the parallel composition of the `Water-level1` component and the `Controller1` component.

Parallel composition of the controller with the Water-level The inputs (resp. outputs) of the `Water-tank1` component are the union of the inputs (resp. outputs) of the `Water-level1` component and the `Controller1` component. The assumptions A_{wt_1} (resp.

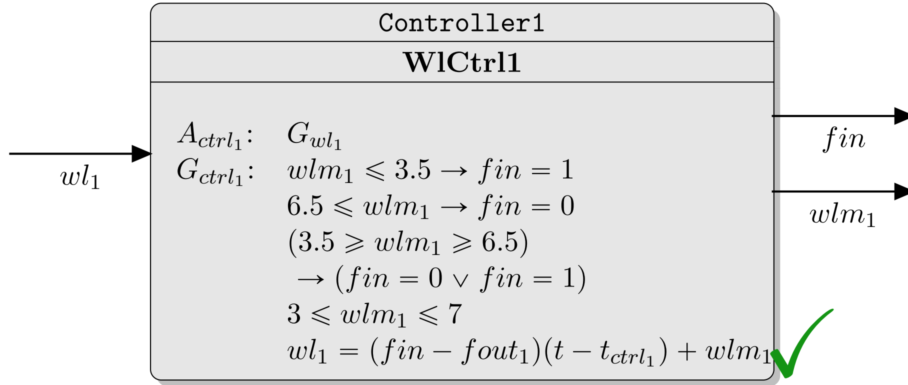


Figure 3.23: Graphical representation of the `Controller1` component

guarantees G_{wt_1}) of the `Water-tank1` are the conjunction of A_{wl_1} and A_{ctrl_1} (resp. G_{wl_1} and G_{ctrl_1}). The behavior is detailed in Definition 31.

A summary of the parallel composition is given in the Figure 3.24. Notice that for now, we have a proof of satisfaction of their contract by the components `Water-level1` and `Controller1`. We do not still have a proof that the component `Water-tank1` satisfies its contract.

Behavior of `Water-tank` The behavior **WT1** of the `Water-tank` component is presented in the Definition 31.

Definition 31 (Water-tank behavior)**.** *The behavior **WT1** of the first tank results from the parallel composition of **Wl1** and **WlCtrl1**.*

$$\begin{pmatrix} \left(?t \leqslant t_{ctrl_1} + \delta_1; wlm_1 := wl_1; (?wlm_1 \geqslant 6.5; fin := 0) \cup (?wlm_1 \leqslant 3.5; fin := 1); t_{ctrl_1} := t\right) \\ \cup\left(\dot{wl_1} = fin - fout_1, \dot{t} = 1 \ \ \& \ \ (wl_1 \geqslant 0 \wedge t \geqslant 0 \wedge \delta_1 \geqslant t - t_{ctrl_1} \geqslant 0 \wedge t \geqslant t_{ctrl_1})\right) \end{pmatrix}^*$$

The addition of temporal constraints makes sense once we have composed the components. The variable t_{ctrl_1} remembers the instant of the last execution of the controller. It is then equal to t. The water-level evolves along the time, and the difference between t and t_{ctrl_1} grows until it reaches δ_1. Then we know that the controller has to execute.

Application of the composition theorem The conditions of the Theorem 4 are respected. The outputs of both components are separated. The guarantees G_{wl_1} of the `Water-level1` do not refer to the outputs of the `Controller1` and the inverse is also true. The guarantee $wl_1 = (fin - fout_1)(t - t_{ctrl_1}) + wlm_1$ contains occurrences of outputs of both components, but we have proved that it is a guarantee of both component. It is thus not a problem to the application of the theorem. The guarantees G_{wl_1} implies the assumptions A_{ctrl_1} trivially since they correspond, and the inverse is true. We have thus automatically the proof of satisfaction of the conjunction of contracts by the resulting component thanks to the application of the theorem.

We have sum up in the Figure 3.25 the `Water-tank1` component. We have now a proof of satisfaction of its contract thanks to the application of the Theorem 4.

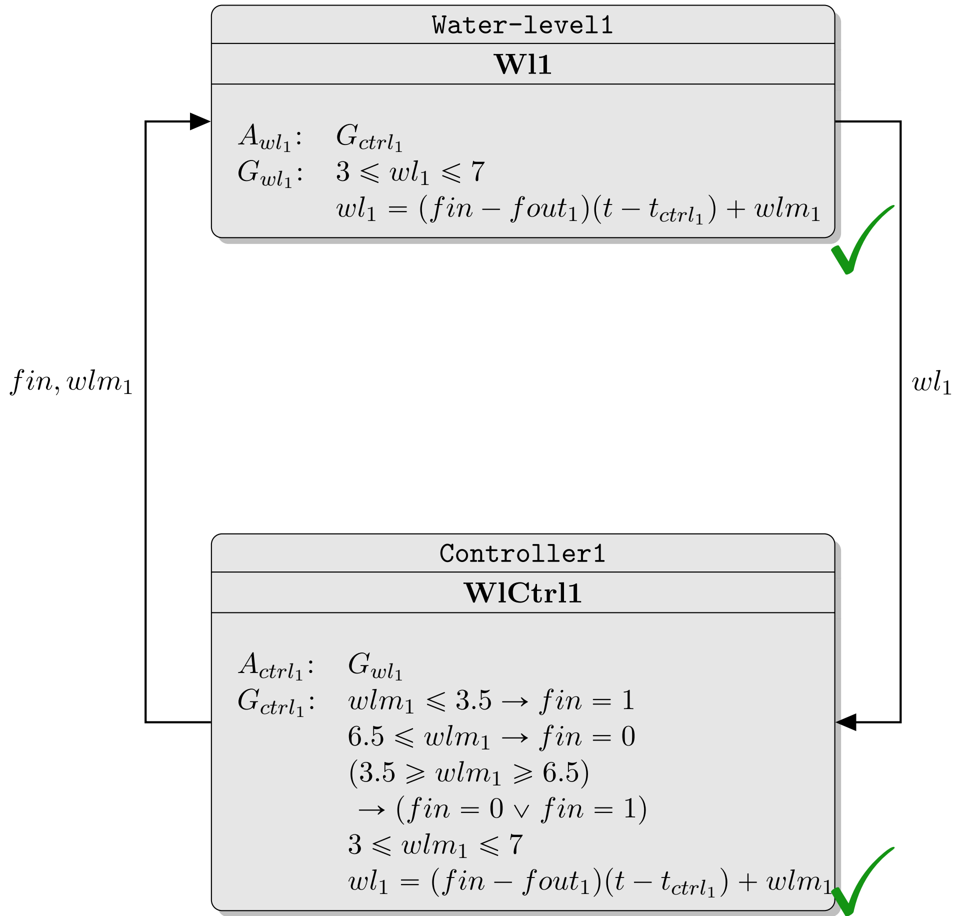


Figure 3.24: Parallel composition of `Water-level1` with the `Controller1`

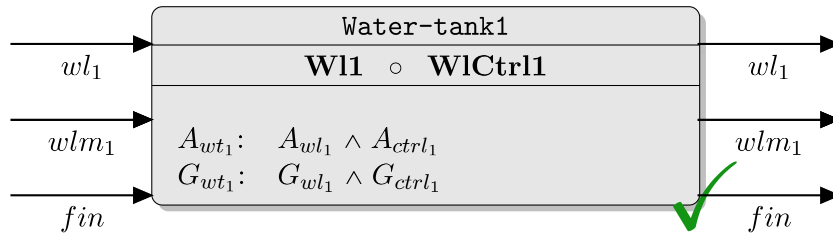


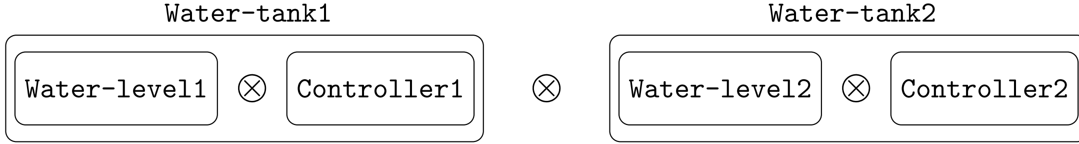
Figure 3.25: Graphical representation of the `Water-tank1` component

Implementation in KeYmaera X We have implemented in KeYmaera X the component resulting from the composition 5.3.2. At the end of the file, in commentary, there is the tactic used to achieve the proof. We have followed strictly our procedure detailed in the proof of the Theorem 4. It requires a minimal supplementary proof effort from the proof of each component. It may be automated.

The water-tank file shows us that even a simple example quickly grows in size. Our approach requires that the engineer only consider the basic component, here the water-level and the controller, which are much simpler to understand and prove.

3.5.5 Water-Plant

We compose two water-tanks with our parallel composition operator. It follows an intuitive design in the sense that we first compose the `Water-level1` component with the `Controller1` component to obtain a first tank `Water-tank1`, then compose it with a second water-tank obtained by similar mean.



Second water-tank The second water-tank is a lot similar to the first one, but it is the outlet valve that can be closed or open by the controller, the inlet flow is assumed to be constant. A detailed .kyx file are given for the second water-level 5.3.2, the second controller 5.3.2 and the second water-tank resulting from their composition 5.3.2.

We have summed up in the Figure 3.26.

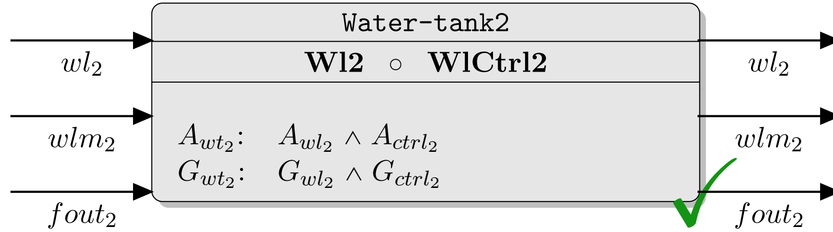


Figure 3.26: Graphical representation of the `Water-tank2` component

Parallel composition The inputs (resp. outputs) of the `Water-plant` component are the union of the inputs (resp. outputs) of the first `Water-tank` component and the second `Water-tank` component. The assumptions A_{wp} (resp. guarantees G_{wp}) of the `Water-plant` are the conjunction of A_{wt_1} and A_{wt_2} (resp. G_{wt_1} and G_{wt_2}). The behavior **WP** of the `Water-plant` component is presented in the Definition 32.

Definition 32 (Water-plant behavior)**.** *The behavior **WP** of the water-plant results from the parallel composition of **WT1** and **WT2**, where **WT1** results from the parallel composition of **Wl1** and **WlCtrl1** and **WT2** results from the parallel composition of **Wl2** and **WlCtrl2**.*

$$\begin{aligned} \boldsymbol{WP} &\triangleq \boldsymbol{WT1}_1 \circ \boldsymbol{WT2} \\ &\triangleq (\boldsymbol{Wl1} \circ \boldsymbol{WlCtrl1}) \circ (\boldsymbol{Wl2} \circ \boldsymbol{WlCtrl2}) \end{aligned}$$

$$\triangleq \left(\begin{array}{l} \big((?t \leqslant t_{ctrl_1} + \delta_1; wlm_1 := wl_1; \\ (?wlm_1 \geqslant 6.5; fin := 0) \cup (?wlm_1 \leqslant 3.5; fin := 1); t_{ctrl_1} := t\big) \\ \cup \ \big(?t \leqslant tctrl_2 + \delta_2; wlm_2 := wl_2; \\ (?wlm_2 \geqslant 6.5; fout_2 := 1) \cup (?wlm_2 \leqslant 3.5; fout_2 := 0); tctrl_2 := t\big) \\ \cup \big(\dot{wl_1} = fin - fout_1, \dot{wl2} = fout_1 - fout_2, \dot{t} = 1 \\ \ \& \ (wl_1 \geqslant 0 \wedge t \geqslant 0 \wedge \delta_1 \geqslant t - t_{ctrl_1} \geqslant 0 \wedge t \geqslant t_{ctrl_1} \\ \wedge wl_2 \geqslant 0 \wedge \delta_2 \geqslant t - tctrl_2 \geqslant 0 \wedge t \geqslant tctrl_2))\big)^* \end{array} \right)^*$$

In particular, we have an example of the parallel composition of two continuous components with the water-levels composed in parallel.

Application of the theorem The conditions of the Theorem 4 are respected. The outputs of both components are separated. The guarantees G_{wt_1} of the `Water-tank1` do not refer to the outputs of the `Water-tank2` and the inverse is also true. The sequent $\mathcal{E}_{wp}, A_{wt_2} \vdash \forall^{wt_1}(G_{wt_1} \rightarrow A_{wt_2})$ is valid since the outputs of the first water-tank are separated from the variables in A_{wt_2}. The inverse is also true. We have thus automatically the proof of satisfaction of the conjunction of contracts by the resulting component thanks to the application of the theorem. It is summed up in the Figure 3.27.

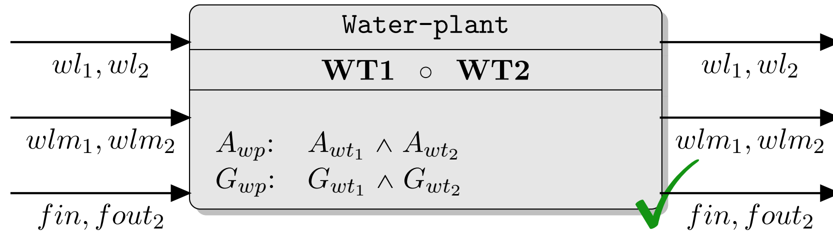


Figure 3.27: Graphical representation of the `Water-plant` component

Implementation in KeYmaera X We have implemented in KeYmaera X the component resulting from the composition 5.3.2. At the end of the file, in commentary, there is the tactic used to achieve the proof. We have followed strictly our procedure detailed in the proof of the Theorem 4. It requires a minimal supplementary proof effort from the proof of each component. It may be automated.

3.5.6 Discussion

Monolithic approach VS Modular approach The source files 5.3.2 show that the design of even a simple water-plant may result in a model quite complex to model. The proof is also difficult to automate. New improvements in KeYmaera X can drastically improve the readability and help to scale, but the same problem will arise for system more complex.

We strongly think that our component-based approach allows to handle both design and proof complexity of systems made up of simple and repetitive parts.

Improvement points Our two examples, the cruise-control and the water-tank shares the same structure : a continuous evolution (the plant) is monitored by a discrete controller. It

is a frequent design in Cyber-physical Systems and is referred to as Computer-Controlled Systems. It includes most of the industrial systems.

In both examples, we have to add a machinery to ensure that the controller executes periodically. This addition is not a function problem, but an architecture problem and the designer should have tools to automate it. It is the subject of Section 4.1.

When we compose several CCS together as in our water-plant Example 32, the controller might be executed on the same computation unit and therefore their execution period may be modified by the parallel execution. We answer to this problem in Section 4.2.

A common design in critical system is the addition of an emergency mode to the nominal mode. The natural way to express it in our framework is to compose in parallel the emergency mode with the nominal mode. But we have to be careful since they have the same outputs. We present a solution in Section 4.3.

When designing such system, one may want to define a causal relation between components. We propose in Section 4.4 the definition of a causal composition operator compatible with the previously defined framework.

Chapter 4

Extensions of our framework

We have presented a parallel composition operator in Chapter 3. The water-plant example in Section 3.5 shows that our approach still requires that the engineer have to reason on non-functional problems to achieve its design and proof. We present in this chapter adaptations of the parallel composition operator to remediate to the problems identified in Subsection 3.5.6.

In Sections 4.1 and 4.2, we show how to adapt the results of the previous Chapter 3 to systematically model Computer-Controlled Systems (CCS) in a modular approach. The idea is to analyse the control period of the plant, *i.e.* the period of time it can evolve without intervention of the controller, and the reactivity of the controller, *i.e.* the execution period of the controller. We retain the associativity property and the ability to retain contracts through composition. To clearly separate the design part of a system and its representation in $d\mathcal{L}$, we continue to use the textual representation of components.

In Section 4.3, we show how to express modes in a systematic and modular way by adapting the parallel composition operator $\circ$.

An usual decomposition of a controller is `Sensor -> Program -> Actuator`. But the three components are not in parallel, but in sequence. In Section 4.4, we define a causal composition operator by adapting our parallel composition operator to enforce an order. It features associativity, essential for modularity, and a theorem allowing to transfer automatically guarantees of each component to the resulting compound component under relaxed conditions.

4.1 Computer-Controlled Systems

We present a component-based approach to model and verify Computer-Controlled Systems (CCS). We aim to simplify the reasoning on such systems and provide methodological insights for designers and proof engineers.

We adapt our previous work by taking into account the period of execution of the controller. We describe the integration and show that it cover the standard encoding of CCS in $d\mathcal{L}$. We adapt the composition Theorem 4 to show that we retain contracts through composition of components. We exemplify it with the water-tank example.

4.1.1 Modeling Computer-Controlled Systems

We present how we model Computer-Controlled System. A CCS is classically composed of a *controller* and a *plant* as in Figure 4.1. The former regulates the behavior of the latter through an actuation. For example, the controller in the water-tank example regulates the water-level by opening or closing a faucet.

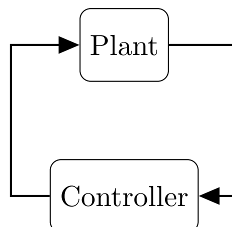


Figure 4.1: Structure of a CCS

The key trait of CCS is the periodic execution of the controller to regulates the controller.

We associate a periodic value Δ_{Ctrl} (resp. Δ_{Plant}) to the controller (resp. the plant). For the control part, it is the *execution period*, *i.e.* the maximal allowed time between two executions of the controller. For the plant, it is the duration in which it can freely evolve and still satisfies the safety property, the *control period*. We add timing constraints to model this requirement.

We use the water-tank example of Section 3.5, except that we distinguish the execution period from the control period. These two notions were conflated in our previous example.

Time. We assume that the passing of time is accessible for every component through the differential equation $\dot{t} = 1 \ \& \ t \geqslant 0$. The variable t represents the time passing linearly and is a global read-only variable. It is initialized to 0.

Controller. We assume that a designer provides the functional behavior of a controller as a discrete program *Ctrl* and the associated *period* Δ_{Ctrl}. The functional behavior represents one execution of the controller. We add timing constraints to model the periodic execution of the controller.

The controller acts every Δ_{Ctrl} units of time. To ensure this periodic execution, we use a *fresh* variable t_{ctrl}. It works as a time-stamp saying that *Ctrl* has executed. We add at the end the assignment $t_{ctrl} := t$, which signals that the controller has executed. We add the guard $?t \leqslant t_{ctrl} + \Delta_{Ctrl}$ before the controller *Ctrl*, which requires that at most Δ_{Ctrl} units of time have passed since the last execution. It results in the following canonical structure for the controller, which is referred later as the *behavior* of the controller.

Definition 33 (Controller)**.** *The* canonical structure *of a controller is given by the hybrid program:*

$$\boldsymbol{Ctrl} \triangleq (?t \leqslant t_{ctrl} + \Delta_{Ctrl}; Ctrl; t_{ctrl} := t)^*$$

This definition allows Zeno behaviors, *i.e.* repetitive executions of the controller without having the continuous systems to evolve. It is a problem when we want to implement a real controller, but we are here interested into the verification of safety properties. Having a more general modeling is thus not a problem. We could forbid this Zeno behavior by adding the test $t_{ctrl} < t$ in the guard. Another possibility is to replace the test $t \leqslant t_{ctrl} + \Delta_{Ctrl}$ with the test $t = t_{ctrl} + \Delta_{Ctrl}$. It assumed here that the controller executes every Δ_{Ctrl} units of time which is unrealistic if we want to obtain a real implementation later.

For example, consider the controller for the water-level of a tank in a water-plant. If the level reaches a maximum (resp. minimum) threshold, here 6.5 (resp. 3.5), then we close the inlet faucet fin (resp. we open the inlet faucet).

Example 35 (Water-level controller)**.** *The* functional behavior *is :*

$$\boldsymbol{Body}_{\mathrm{WlCtrl}} \triangleq wlm := wl; \big((?wlm \geqslant 6.5; fin := 0) \cup (?wlm \leqslant 3.5; fin := 1)\big)$$

It describes how the component operates, but does not refer to structural constraints like the execution period. The period Δ_{wlCtrl} of the controller is 0.05 sec. We augment the functional behavior $\boldsymbol{Body}_{\mathrm{WlCtrl}}$ with the period to obtain the behavior ***WlCtrl**:*

$$\begin{pmatrix} ?t \leqslant t_{ctrl} + 0.05; \\ wlm := wl; (?wlm \geqslant 6.5; fin := 0) \cup (?wlm \leqslant 3.5; fin := 1); \\ t_{ctrl} := t \end{pmatrix}^*$$

At most every 0.05 seconds, the water-level controller samples the current water-level and decides accordingly to open or close the inlet valve.

Retro-action It is possible to take account of the retro-action. Instead of using the last sensing in the modeling, we use the previous one, *i.e.* the sensing performed Δ_{Ctrl} units of time before. We use a temporary variable to memorize it. It accurately models the fact that the actuation of controller at a time t_0 is in function of the sensing at $t_0 - \Delta_{Ctrl}$.

Example 36 (Water-tank controller with retro-action)**.** *The behavior of the water-tank controller with retro-action is:*

$$\left(\begin{array}{l} ?t \leqslant t_{ctrl} + \Delta_{Ctrl}; \\ prewlm := wlm; wlm := wl; \\ (?prewlm \geqslant 6.5; fin := 0) \cup (?prewlm \leqslant 3.5; fin := 1); \\ t_{ctrl} := t \end{array}\right)^*$$

In the example, the level of water of the tank is assigned to the variable wlm, then to the variable $prewlm$. wlm is here to stock the value for one more iteration of the controller and simulates a retro-action.

Plant. We assume that a designer provides the functional behavior of the *plant* as a differential equation $ODE \ \& \ H$ and the *control period* Δ_{Plant}. We implement it by adding the formula $t \leqslant \Delta_{Plant}$ in the evolution domain.

Definition 34 (Plant)**.** *The* canonical structure *of a plant is:*

$$\textbf{\textit{Plant}} \triangleq ODE, \dot{t} = 1 \ \& \ H \wedge t \leqslant \Delta_{Plant}$$

The addition of the formula $t \leqslant \Delta_{Plant}$ adequately models that we consider an evolution of the system in the duration of the control period.

The value of the control period $\Delta_{Plant}$is obtained by external methods and is not computed by our methodology. In practice, it is mostly obtained from the knowledge of engineers of the field with some tolerance.

The evolution of the water-level is the difference between the inlet flow fin and the outlet flow $fout$. The water-level is assumed to always be positive.

Example 37 (Water-level)**.** *The* functional behavior *of the water-level is the following differential equation:*

$$\dot{wl} = fin - fout \ \& \ wl \geqslant 0$$

The control period Δ_{wl} *of the water-level is* 0.2 *sec. The* behavior ***Wl*** *of the water-level component is thus given by:*

$$\dot{wl} = fin - fout, \dot{t} = 1 \ \& \ wl \geqslant 0 \wedge t \leqslant 0.2$$

Our proposed modeling is sufficient to reason on the system. Indeed, when we compose the plant with the controller, we iterate the interaction. Thus, considering only the evolution on the duration Δ_{Plant} amounts to consider one step of the loop Plant-Controller.

Full system The full system is obtained by applying our parallel composition operator on the plant and the controller with a slight difference.

Definition 35 (Computer-Controlled System)**.** *The* canonical structure *of the behavior of a Computer-Control System is:*

$$\boldsymbol{CCS} \triangleq \big((Plant, \dot{t} = 1 \ \& \ H \wedge 0 \leqslant t - t_{ctrl} \leqslant \Delta_{Ctrl}) \\ \cup(?t \leqslant t_{ctrl} + \Delta_{Ctrl}; Ctrl; t_{ctrl} := t)\big)^*$$

The controller and the plant interact by the use of the variable t_{ctrl}. At the beginning of one loop iteration, we have $t = t_{ctrl}$ and thus $t - t_{ctrl} = 0$. The difference $t - t_{ctrl}$ grows along the evolution of time until the point $t - t_{ctrl} = \Delta_{Ctrl}$. Then, the controller has to act, but it can act before.

The difference with the parallel composition operator in Definition 22 is that we replace the formula $t \geqslant \Delta_{Plant}$ by the formula $0 \leqslant t - t_{ctrl} \leqslant \Delta_{Ctrl}$. We detail in the proof of the Theorem 6 why this replacement is safe.

For our system to fit the desired modeling, we must have $\Delta_{Ctrl} \leqslant \Delta_{Plant}$, *i.e.* that the reactivity of the controller is shorter to the control period of the plant. Otherwise, there may be runs of the whole system where the controller can not execute and the system is stuck. Yet, we do not forbid such composition, but we will not be able to retain the contracts through composition.

Example 38 (Water-tank)**.** *We compose the water-level with the water-level controller to obtain the water-tank system. The behavior* ***Water-tank*** *resulting from the composition of the water-level and the water-level controller is:*

$$\left(\begin{array}{l} \quad (\dot{wl} = fin - fout \ \& \ wl \geqslant 0 \wedge 0 \leqslant t - t_{ctrl} \leqslant 0.05) \\ \cup \ (?t \leqslant t_{ctrl} + 0.05; wlm := wl; (?wlm \geqslant 6.5; fin := 0) \cup (?wlm \leqslant 3.5; fin := 1); t_{ctrl} := t) \end{array} \right)^*$$

The composition is safe because the period of the controller ($\Delta_{WlCtrl} = 0.05$) is inferior to the control period of the plant ($\Delta_{wl} = 0.2$).

We retain the general structure of component in Chapter 3 which is essential to obtain modularity. Yet, one may ask if we accurately capture all CCS systems. We show in the next subsection that we capture all systems that are encoded by the standard encoding of CCS that we repeatedly found in the literature in $d\mathcal{L}$. It gives us confidence that our proposed approach captures such systems.

4.1.2 Coverage of the standard encoding

We first define what we call the *standard encoding* of Computer-Controlled Systems in $d\mathcal{L}$. It is a recurrent structure in the literature. We then show that our modeling proposition can model every system that can be modeled with the standard encoding.

Definition 36 (Standard encoding)**.** *The* standard encoding *of a CCS is:*

$$(Ctrl; t_{ctrl} = t;\ Plant, \dot{t} = 1 \ \& \ H \wedge 0 \leqslant t - t_{ctrl} \leqslant \Delta_{Ctrl})^*$$

The alternation of the controller and the plant is obtained by the use of the sequence operator ; . Notice that a similar encoding is used to represent the cyclic relation to time. We have a notification variable t_{ctrl} that is reset to t after the execution of the controller. This modeling is simple, yet very efficient in a monolithic approach. But we think that it is not the appropriate approach if we want to use a component-based approach.

Remember that $\rho(\alpha)$, where α is an hybrid program, is the set of *reachable states* of α (cf Definitions 2 and 4). We show that the set of reachable states of the standard encoding of CCS is included in the set of reachable states of our encoding. It means that a system that can be modeled with the standard encoding can also modeled with our approach.

Theorem 5 (Coverage of the standard encoding)**.** *Every system that can be modeled with the standard encoding may be modeled with our encoding, i.e.*

$$\begin{array}{ll} & \rho\big((Ctrl; t_{ctrl} = t;\ Plant, \dot{t} = 1\ \&\ H \wedge 0 \leqslant t - t_{ctrl} \leqslant \Delta_{Ctrl})^*\big) \\ \subseteq & \rho\big(((Plant, \dot{t} = 1\ \&\ H \wedge 0 \leqslant t - t_{ctrl} \leqslant \Delta_{Ctrl}) \cup (?t \leqslant t_{ctrl} + \Delta; Ctrl; t_{ctrl} := t))^*\big) \end{array}$$

Proof. We prove the equivalence by unfolding the semantic definition of our proposed encoding and show that one of the subset of reachable states of our encoding matches exactly the set of reachable states of the standard encoding. First, we add the test $?t \leqslant t_{ctrl} + \Delta_{Ctrl}$ in front of the standard encoding. It does not change the reachability, but is here to simplify the identification to our encoding. We have the following equality.

$$\begin{array}{ll} & \rho\big((Ctrl; t_{ctrl} = t;\ Plant, \dot{t} = 1\ \&\ H \wedge 0 \leqslant t - t_{ctrl} \leqslant \Delta_{Ctrl})^*\big) \\ = & \rho\big((?t \leqslant t_{ctrl} + \Delta_{Ctrl}; Ctrl; t_{ctrl} = t;\ Plant, \dot{t} = 1\ \&\ H \wedge 0 \leqslant t - t_{ctrl} \leqslant \Delta_{Ctrl})^*\big) \end{array}$$

The right-hand side is trivially included in the left-hand side since we just add information, and thus potentially reduces the set of worlds. For the other inclusion, if it is not true, then there is a world w in $\rho\big((Ctrl; t_{ctrl} = t;\ Plant, \dot{t} = 1\ \&\ H \wedge 0 \leqslant t - t_{ctrl} \leqslant \Delta_{Ctrl})^*\big)$ such that $w \not\models t \leqslant t_{ctrl} + \Delta_{Ctrl}$, but it contradicts the assumption that the continuous behaviors evolve in a world where $t \leqslant t_{ctrl} + \Delta_{Ctrl}$ is true. We have thus the other inclusion.

We have to prove:

$$\begin{array}{ll} & \rho\big((?t \leqslant t_{ctrl} + \Delta_{Ctrl}; Ctrl; t_{ctrl} = t;\ Plant, \dot{t} = 1\ \&\ H \wedge 0 \leqslant t - t_{ctrl} \leqslant \Delta_{Ctrl})^*\big) \\ \subseteq & \rho\big(((Plant, \dot{t} = 1\ \&\ H \wedge 0 \leqslant t - t_{ctrl} \leqslant \Delta_{Ctrl}) \cup (?t \leqslant t_{ctrl} + \Delta; Ctrl; t_{ctrl} := t))^*\big) \end{array}$$

We adopt the following abbreviations:

$$\begin{array}{ll} P & = Plant, \dot{t} = 1\ \&\ H \wedge 0 \leqslant t - t_{ctrl} \leqslant \Delta_{Ctrl} \\ C & = ?t \leqslant t_{ctrl} + \Delta_{Ctrl}; Ctrl; t_{ctrl} := t \end{array}$$

Thus, the standard encoding corresponds to $(C; P)^*$ and our encoding corresponds to $(C \cup P)^*$. We unfold the semantic definition of the iteration. We obtain the following equality:

$$\begin{array}{ll} & \rho\big((C \cup P)^*\big) \\ = & \bigcup_{n \in \mathbb{N}} \rho\big((C \cup P)^n\big) \end{array}$$

We have thus $\rho\big((C \cup P)^2\big) \subseteq \rho\big((C \cup P)^*\big)$.
We unfold $\rho\big((C \cup P)^2\big)$:

$$\begin{array}{ll} & \rho\big((C \cup P)^2\big) \\ = & \rho\big((C \cup P); (C \cup P)\big) \\ = & \rho(C; C) \cup \rho(C; P) \cup \rho(P; C) \cup \rho(P; P) \end{array}$$

Thus the left-hand side is included in the right-hand side.

□

4.1.3 Modular proof of a Computer-Controlled System

The Subsection 4.1.1 have presented how to modularly model a CCS. We want now to be able to still retain contracts through composition. This subsection presents an adaptation of the Theorem 4 to transfer the respective contract of the plant and the controller through composition.

Environment. We assume that the designer provides a description of the environment, which is denoted by $\mathcal{E}$. It contains all the variables that are not outputs of a component. They cannot be controlled and are exterior to the considered system. A classical example in hybrid systems is the gravity value g. The environment also regroups the initial values of the system.

Example 39 (Water-tank environment)**.** *In the water-tank, the environment $\mathcal{E}_{wt}$ is the* outlet flow $fout$ *which is equal to* 0 *or* 1*. There is also the control period Δ_{Ctrl} of the plant which is* 0.2 *sec and the period Δ_{Ctrl} of the controller which is* 0.05*. The initial value of the water-level wl and the measured water-level wlm are* 5*. The inlet valve fin is closed at the initialization.*

$$\mathcal{E}_{wt} \triangleq fout = 0.75 \wedge \Delta_{wlCtrl} = 0.05 \wedge \Delta_{wl} = 0.2 \wedge wl = wlm = 5 \wedge fin = 0$$

Controller We assume that the designer provides the assumptions A_{Ctrl} and guarantees G_{Ctrl} on the controller. Following the restriction of the Theorem 4, guarantees must not refer to outputs of the plant. Remember that the controller is said to satisfy its contract (Definition 20) if the following formula is valid:

$$(\mathcal{E} \wedge A_{Ctrl}) \rightarrow [\mathbf{Ctrl}]G_{Ctrl}$$

We prove the validity by using the sequent calculus of $d\mathcal{L}$, implemented in KeYmaera X.

Example 40 (Contract of the water-level controller)**.** *From the Example 3.22, the contract of the water-level controller is:*

$$\begin{cases} A_{wlCtrl}: & G_{wl} \\ G_{wlCtrl}: & wlm \leqslant 3.5 \rightarrow fin = 1 \\ & 6.5 \leqslant wlm \rightarrow fin = 0 \\ & (3.5 \geqslant wlm \geqslant 6.5) \rightarrow (fin = 0 \vee fin = 1) \\ & wl = (fin - fout)(t - t_{ctrl}) + wlm \end{cases}$$

We have already a proof of the sequent:

$$\mathcal{E}_{wt}, A_{wlCtrl} \vdash [\boldsymbol{WlCtrl}]G_{wlCtrl}$$

Plant Similarly to the controller, we assume that the designer provides assumptions A_{Plant} and guarantees G_{Plant} on the plant. Again, the guarantees of the plant must not refer to the outputs of the controller. The plant satisfies its contract if the following formula is valid:

$$(\mathcal{E} \wedge A_{Plant}) \rightarrow [\mathbf{Plant}]G_{Plant}$$

We prove the validity by using the sequent calculus of $d\mathcal{L}$, implemented in KeYmaera X.

Example 41 (Contract of the water-level)**.** *From the Example 3.20, the contract of the water-level is:*

$$\begin{cases} A_{wl}: & G_{wlCtrl} \\ G_{wl}: & 3 \leqslant wl \leqslant 7 \\ & wl = (fin - fout)(t - t_{ctrl}) + wlm \end{cases}$$

We have already a proof of the sequent:

$$\mathcal{E}_{wt}, A_{wl} \vdash [\boldsymbol{Wl}]G_{wl}$$

Full system The contract for the full system is the conjunction of the assumptions and of the guarantees. As in Section 3.3, we want that the system **CCS** as defined in Definition 35 satisfies the conjunction of contracts $(A_{Ctrl} \wedge A_{Plant}, G_{Ctrl} \wedge G_{Plant})$. We adapt the Theorem 4 by adding the condition that the execution period must be shorter than the control period, *i.e.* $\Delta_{Ctrl} \leqslant \Delta_{plant}$, to retain contracts.

Theorem 6 (Safe composition of a Plant and a Controller)**.** *Let Plant and Ctrl be two behaviors as defined previously with respective contracts* (A_{Plant}, G_{Plant}) *and* (A_{Ctrl}, G_{Ctrl}). *Assume that we have two proof trees of* $\mathcal{E}, A_{Plant} \vdash [Plant]G_{Plant}$ *and* $\mathcal{E}, A_{Ctrl} \vdash [Ctrl]G_{Ctrl}$ *respectively, where* $\mathcal{E}$ *is the environment. Furthermore, assume that*

(a) $BV(Plant) \cap BV(Ctrl) = \varnothing$,

(b_1) $BV(Plant) \cap FV(G_{Ctrl}) = \varnothing$,

(b_2) $BV(Ctrl) \cap FV(G_{Plant}) = \varnothing$ *and* $BV(Ctrl) \cap FV(H)$,

(c_1) $A_{Plant} \vdash \forall^{Ctrl}(G_{Ctrl} \rightarrow A_{Plant})$,

(c_2) $A_{Ctrl} \vdash \forall^{Plant}(G_{Plant} \rightarrow A_{Ctrl})$,

(d) $\Delta_{Ctrl} \leqslant \Delta_{plant}$.

Then it exists a proof tree of $\mathcal{E}, A_{Plant}, A_{Ctrl} \vdash [\boldsymbol{CCS}](G_{Plant} \wedge G_{Ctrl})$.

The first assumption (a) assumes components to have separate internal variables and requires them to define disjoint output variables (i.e. unique definitions), which essentially amounts to good modeling practice. The second assumption (b_1) (resp. (b_2)) requires the safety property G_{Ctrl} (resp. G_{Plant}) to guard the behavior of the system α (resp. β), i.e. its outputs, and of course not βs (resp. α). It hence seems natural to require its separation with G_{Plant} (resp. G_{Ctrl}). The condition (c_1) (resp. (c_2)) requires the assumptions A_β (resp. A_α) to be implied by the guarantees G_{Ctrl} (resp. G_{Plant}).

Proof. We re-use the result of the Theorem 4. In order to do so, we have to justify our slight change where we replace the formula $t \leqslant \Delta_{Plant}$ with the formula $0 \leqslant t - t_{Ctrl} \leqslant \Delta_{Ctrl}$ in the Definition 35.

First, the condition (d) tells us that $\Delta_{Ctrl} \leqslant \Delta_{Plant}$. Thus, if the sequent $\mathcal{E}, A_{Plant} \vdash [Plant, \dot{t} = 1 \ \& \ H \wedge t \leqslant \Delta_{Plant}]G_{Plant}$ is valid, we have that the sequent $\mathcal{E}, A_{Plant} \vdash [Plant, \dot{t} = 1 \ \& \ H \wedge t \leqslant \Delta_{Ctrl}]G_{Plant}$ is valid too. It shorten the amount of time that the plant is allowed to evolve. So if we have proved that it satisfies the guarantees for a defined amount of time, we have trivially that it satisfies guarantees for a smaller amount of time.

We can then safely replace the formula $t \leqslant \Delta_{Ctrl}$ by $0 \leqslant t - t_{ctrl} \leqslant \Delta_{Ctrl}$ in the evolution domain since it represents the same amount of time. It is now possible to apply the Theorem 4 to conclude. □

Example 42 (Contract of the water-tank)**.** *From the Example 3.25, the contract of the water-tank resulting from the composition of the water-level and the controller is:*

$$\begin{cases} A_{wt}: & A_{wl} \wedge A_{wlCtrl} \\ G_{wt}: & G_{wl} \wedge G_{wlCtrl} \end{cases}$$

The conditions of the Theorem 6 are satisfied. The conditions (*a*) *to* (*c*2) *are already considered in the previous section. Plus, we have that* $\Delta_{wlCtrl} \leqslant \Delta_{wl}$ *We have a proof of the following sequent:*

$$\mathcal{E}, A_{wt} \vdash [\textit{\textbf{Water-tank}}]G_{wt}$$

Conclusion

We have presented how we can adapt our parallel composition operator to modularly model and prove a Computer-Controlled System and reason on a non-functional property like the execution period and the control period. But the presentation in this section is only for two elements, and we want to extend this approach with to a plant where several systems Plant-Controller are composed in parallel. The next section is devoted to the presentation of parallel composition with a systematic integration of timed constraints.

4.2 Parallel composition in a timed framework

We want to extend the integration of temporal considerations for every component in a timed framework. Reasoning on temporal executions of CPS is an important step during the design of such systems and this extension aims to ease its development.

When we execute two programs in parallel on one CPU, their execution periods, that we identify to Worst Case Execution Time (WCET), is augmented. The computation resources may be preempted by the other program and the first program has to wait. More precisely, the resulting execution period is the sum of respective WCET. We have also to take into consideration if the composition with the continuous part is still valid, *i.e.* that the resulting execution period is still inferior to the control period of the plant.

The parallel composition operator in Definition 22 is too general to handle this kind of reasoning. We present in the next section its adaptation by extending the idea of Section 4.1 and exemplify it with the water-plant example. We retain the algebraic properties, commutativity and associativity, as well as the theorem guaranteeing that the conjunction of contracts is preserved through composition.

In Subsection 4.2.1, we defined how we integrate systematically the notion of execution periods and control period to obtain a so-called timed component. The Subsection 4.2.2 details the parallel composition between discrete timed components and we show that we retain associativity and the composition theorem. The Subsection 4.2.3 is similar, but for continuous timed component. The Subsection 4.2.4 show how we associate a discrete component and a continuous component, where each of them may result from previous compositions. It is a generalization of Section 4.1. Finally, the Subsection 4.2.5 show how to aggregate all the previous definitions to obtain a parallel composition of timed general components.

4.2.1 Definition of a timed component

In this subsection, we present the definition of a component with the timed information and how it is translated in $d\mathcal{L}$. The designer should only provide the textual representation and the translation in the $d\mathcal{L}$ framework is handled automatically.

We associate to every atomic component a duration which is assumed to be provided by the designer. For the discrete case, it is called the *execution period* and is understood as the time the component takes to execute. We may assimilate it to WCET. In the continuous case, it is the *control period* of the continuous system, *i.e.* the duration during which the continuous component can evolves freely without an intervention of a controller. We extend the definition by adding tests and assignment in a similar manner that in Section 4.1.1. To keep the structure of components and the associated properties, we add tests and assignment to discrete atomic sub-component along with the execution period provided by the designer. The execution period of a compound component is obtained through composition of atomic components.

Textual representation For a discrete component, Δ_α represents the *execution period* of the component. We just add this information to our previously defined textual representation in Subsection 3.1.1. For example, for the first water-level controller, we add that the execution period $\Delta_{WlCtrl1}$ is 0.03 seconds.

Example 43 (Textual representation of water-level controller). *From the Example 3.22, the* water-level controller *of the first tank is defined by:*

`Component: Controller1`
`Period:`
$\Delta_{WlCtrl1} \;=\; 0.03$
`Inputs:`
wl_1
`Assumptions:`
G_{wl_1}
`Outputs:`
fin
wlm_1
`Guarantees:`
$3 \leqslant wlm_1 \leqslant 7$
$3.5 \geqslant wlm_1 \rightarrow fin = 1$
$wlm_1 \geqslant 6.5 \rightarrow fin = 0$
$(3.5 \leqslant wlm_1 \leqslant 6.5) \rightarrow (fin = 0 \vee fin = 1)$
$wl_1 = (fin - fout_1)(t - t_{ctrl_1}) + wlm_1$
`Functional behavior:`
$\mathbf{Body}_{WlCtrl1} \triangleq wlm_1 := wl_1;\ (?wlm_1 \geqslant 6.5;\ fin := 0) \cup (?wlm_1 \leqslant 3.5;\ fin := 1)$

We have the same for the water-level controller `WlCtrl2` of the second tank for which the execution $\Delta_{WlCtrl2}$ is 0.02.

For a continuous component, ε_α represents the *control period* of the component. As for the discrete case, we add this information in the textual representation. For example, for the water-level component of the first tank, we have $\varepsilon_{Wl_1} = 0.2$.

Example 44 (Textual representation of water-level)**.** *The* water-level *of the first tank is defined by the following component:*

```
Component: Water-level1
Control period:
```
$\varepsilon_{Wl1} \; = \; 0.2$
```
Inputs:
```
fin
wlm_1
```
Assumptions:
```
G_{ctrl_1}
```
Outputs:
```
wl_1
```
Guarantees:
```
$3 \; \leqslant \; wl_1 \; \leqslant \; 7$
$wl_1 \; = \; (fin - fout_1)(t - t_{ctrl_1}) + wlm_1$
```
Functional behavior:
```
$\dot{wl_1} \; = \; fin \; - \; fout_1 \quad \& \quad wl_1 \; \geqslant \; 0$

where G_{ctrl_1} *are the guarantees of the component* `Controller1`.

We have the same for the water-level `Wl2` of the second tank for which the control period ε_{Wl2} is 0.1.

Behavior of a timed component To every discrete atomic component `A` of functional behavior α, we associate a *fresh* variable t_α which is used to specify the time stamp of the component in an execution cycle.

Definition 37 (Behavior of an atomic component)**.** *Let* `A` *be an atomic component of duration* Δ_α *and functional behavior* α*. If* α *is* discrete*, then the behavior is:*

$$(?t \leqslant t_\alpha + \Delta_\alpha; \alpha; t_\alpha := t)^*$$

If α *is* continuous*, i.e.* $\alpha \triangleq \dot{X} = \theta \;\;\&\;\; H$*, then the behavior is:*

$$\dot{X} = \theta, \dot{t} = 1 \;\;\&\;\; H \wedge 0 \leqslant t \leqslant \varepsilon_\alpha$$

The variable t represents the time and it is assumed to be a read-only variable. We add a guard in front of a discrete functional behavior α to test if Δ_α units of times have passed since the last execution of α. It ensures that the discrete program executes periodically every Δ_α units of time. We add the assignment $t_\alpha := t$ which had to be understood as a time stamp signaling that α has executed.

Given the functional behavior of the `Controller1` and the `Water-level1` provided previously, and their respective execution period $\Delta_{WlCtrl1}$ and ε_{Wl1}, we have the following behaviors.

Example 45 (Behavior of the `Controller1` and the `Water-level1`)**.** *The behavior* ***WlCtrl1*** *of the atomic component* `Controller1` *is:*

$$\left(\; ?t \leqslant t_{Ctrl1} + 0.03; \boldsymbol{Body}_{WlCtrl1}; t_{Ctrl1} := t \; \right)^*$$

*The behavior **Wl1** of the atomic component* `Water-level1` *is:*

$$\dot{wl_1} = fin - fout_1, \dot{t} = 1 \;\;\&\;\; wl_1 \geqslant 0 \wedge 0 \leqslant t \leqslant 0.2$$

We have considered how to integrate temporal considerations on components provided by the engineer. The other way to obtain a component is by parallel composition. The parallel composition is different following that it is between purely discrete components, purely continuous components or a mix of both. We detail the several case in the next subsections.

4.2.2 Timed parallel composition for discrete components

We adapt the parallel composition operator for discrete component by taking into account their respective execution period Δ. We assume that the discrete components are executed on one computation unit (*e.g.* one CPU). The resulting period is the sum of each respective periods since it is the Worst Case Execution Time (WCET) for the interleaving on one CPU.

The problem is different if we dispose of several computation units. It is possible to run two components truly in parallel and the resulting WCET will be the maximum of respective WCET.

Modeling We first define the parallel composition of discrete components. The main difference with the definition of a parallel composition operator in Section 3.2 is the presence of the execution period and it is modified through composition.

Definition 38 (Parallel composition of discrete components)**.** *The component resulting from the parallel composition of two discrete components* `A` *and* `B` *is:*

`Compound Component: AB`
`Period:`
 $\Delta_\alpha \;+\; \Delta_\beta$
`Inputs:`
 `A.inputs` $\cup$ `B.inputs`
`Assumptions:`
 $A_\alpha \;\wedge\; A_\beta$
`Outputs:`
 `A.outputs` $\cup$ `B.outputs`
`Guarantees:`
 $G_\alpha \;\wedge\; G_\beta$
`Functional behavior:`
 $\alpha \;\otimes\; \beta$

There is no need to specify the period, the inputs, the assumptions, the outputs, the guarantees and the behavior of the resulting component, these characteristics are automatically inherited from the composition. The inputs (resp. outputs) are the union of respective inputs (resp. outputs). The assumptions (resp. guarantees) are the conjunction of respective assumptions (resp. guarantees). The parallel composition of discrete components is performed on one CPU, which means interleaving. The resulting period is the sum of respective periods. We explain in the next sections how it unfolds in $d\mathcal{L}$. The behavior is given by the parallel

composition operator $\otimes$ detailed below. We use a different symbol to mark that we are in a timed framework.

As in Definition 22, the timed parallel composition consists of the non-deterministic choice between each behaviors α and β. The parallelism occurs by the interleaving of traces. The only difference is that we replace the occurrences of Δ_α (resp. Δ_β) by $\Delta_{\alpha\beta}$. The execution period of each sub-components is now the execution period of the whole discrete system.

The notation $\alpha_{\Delta_\alpha}^{\Delta_{\alpha\beta}}$ represent the program α where every occurrences of Δ_α is replaced by $\Delta_{\alpha\beta}$. Recall that the general form of a discrete behavior α is given by $(\alpha_1 \cup \ldots \cup \alpha_n)^*$, where α_i are composed of functional behaviors of sub-components of α augmented with the test $?t \leqslant t_{\alpha_i} + \Delta_\alpha$ and the assignment $t_{\alpha_i} := t$.

Definition 39 (Parallel composition of discrete components). *Let α and β be two behaviors of discrete compound components* `A` *and* `B` *with respective periods Δ_α and Δ_β. The parallel composition $\alpha \otimes \beta$ is given by:*

$$\left(\alpha_1{}_{\Delta_\alpha}^{\Delta_{\alpha\beta}} \cup \ldots \cup \alpha_n{}_{\Delta_\alpha}^{\Delta_{\alpha\beta}} \cup \beta_1{}_{\Delta_\beta}^{\Delta_{\alpha\beta}} \cup \ldots \cup \beta_m{}_{\Delta_\beta}^{\Delta_{\alpha\beta}} \right)^*$$

where $\Delta_{\alpha\beta} = \Delta_\alpha + \Delta_\beta$.

For example, we can compose the two water-level controller in parallel.

Example 46 (Composition of two water-level controllers). *We want to compose* `Controller1` *and* `Controller2` *on one CPU. The functional behavior of the first controller is $\boldsymbol{Body}_{WlCtrl1}$ and is $\boldsymbol{Body}_{WlCtrl2}$ for the second controller. The first controller has a period $\Delta_{WlCtrl1}$ of* 0.03 *seconds, and the second has a period $\Delta_{WlCtrl2}$ of* 0.02 *seconds. The behavior resulting from the parallel composition of each respective behaviors is:*

$$\left((?t \leqslant t_{Ctrl1} + 0.05; \boldsymbol{Body}_{WlCtrl1}; t_{Ctrl1} := t) \cup (?t \leqslant t_{Ctrl2} + 0.05; \boldsymbol{Body}_{WlCtrl2}; t_{Ctrl2} := t) \right)^*$$

Algebraic properties We retain the algebraic properties of the parallel composition operator $\circ$ (cf Propositions 3 and 4). The commutativity means that the order of composition is not important. The associativity ensure that we can build a system step-by-step.

Proposition 5 (Commutativity and Associativity). *Let α, β and γ be behaviors of discrete components* `A`, `B` *and* `C`, *and of respective periods Δ_α, Δ_β and Δ_γ.*

$$\begin{array}{lcll} \alpha \otimes \beta & = & \beta \otimes \alpha & \textit{(Commutativity)} \\ (\alpha \otimes \beta) \otimes \gamma & = & \alpha \otimes (\beta \otimes \gamma) & \textit{(Associativity)} \end{array}$$

We retain the two algebraic properties because the sum operation is also commutative and associative. In our definition, it is equivalent to update the period with $\Delta_\alpha + \Delta_\beta$ or $\Delta_\beta + \Delta_\alpha$.

Proof. We first consider the case of commutativity. The behavior resulting from the parallel composition of `A` with `B` is $\big(\alpha_1{}_{\Delta_\alpha}^{\Delta_{\alpha\beta}} \cup \ldots \cup \alpha_n{}_{\Delta_\alpha}^{\Delta_{\alpha\beta}} \cup \beta_1{}_{\Delta_\beta}^{\Delta_{\alpha\beta}} \cup \ldots \cup \beta_m{}_{\Delta_\beta}^{\Delta_{\alpha\beta}}\big)^*$ and the one resulting from the parallel composition of `A` with `B` is $\big(\beta_1{}_{\Delta_\beta}^{\Delta_{\alpha\beta}} \cup \ldots \cup \beta_m{}_{\Delta_\beta}^{\Delta_{\alpha\beta}} \cup \alpha_1{}_{\Delta_\alpha}^{\Delta_{\alpha\beta}} \cup \ldots \cup \alpha_n{}_{\Delta_\alpha}^{\Delta_{\alpha\beta}}\big)^*$.

We have the following equality:

$$\begin{array}{rcl} \Delta_{\alpha\beta} & = & \Delta_\alpha + \Delta_\beta \\ & = & \Delta_\beta + \Delta_\alpha \\ & = & \Delta_{\beta\alpha} \end{array}$$

The behaviors are thus equivalent because of the commutativity of the non-deterministic choice operator $\cup$.

For the associativity, the key point is that the sum operation is associative, *i.e.* that:

$$(\Delta_\alpha + \Delta_\beta) + \Delta_\gamma = \Delta_\alpha + (\Delta_\beta + \Delta_\gamma)$$

The proof is then similar to the proof of associativity of the parallel composition operator $\circ$ in Proposition 4. $\square$

Composition theorem The contract for the timed parallel composition of components `A` and `B` is the conjunction of respective assumptions and guarantees. As in Section 3.3, we want that the component resulting from the parallel composition as in Definition 39 satisfies the conjunction of contracts $(A_\alpha \wedge A_\beta, G_\alpha \wedge G_\beta)$. We adapt the Theorem 4 by adding the condition that the execution period of a component must not occurs in its functional behavior, nor its guarantees, to retain contracts.

Theorem 7. *Let α and β be two behaviors of discrete components `A` and `B` with respective contracts (A_α, G_α) and (A_β, G_β) and respective periods Δ_α and Δ_β. Assume that we have a proof tree of $\mathcal{E}, A_\alpha \vdash [\alpha]G_\alpha$ and $\mathcal{E}, A_\beta \vdash [\beta]G_\beta$. Then, under the conditions:*

- *(a)* $BV(\alpha) \cap BV(\beta) = \varnothing$,
- *(b_1)* $BV(\alpha) \cap FV(G_\beta) = \varnothing$,
- *(b_2)* $BV(\beta) \cap FV(G_\alpha) = \varnothing$,
- *(c_1)* $A_\alpha \vdash \forall^\beta(G_\beta \rightarrow A_\alpha)$,
- *(c_2)* $A_\beta \vdash \forall^\alpha(G_\alpha \rightarrow A_\beta)$,
- *(d) The period Δ_α (resp. Δ_β) does not occur in the functional behavior of `A` or G_α (resp. functional behavior of `B` or G_β).*

We automatically obtain a proof tree of the sequent:

$$\mathcal{E}, A_\alpha, A_\beta \vdash [\alpha \otimes \beta](G_\alpha \wedge G_\beta)$$

The three first conditions are similar to the Theorem 4. The fourth condition forbids the occurrences of period Δ_α in the functional behavior α or the guarantee G_α. It is not restrictive on the design of hybrid systems since a component should not rely on its execution period or guarantee a property related to it. The period of execution of a program in a CPU is extremely difficult to plan and may vary because of others programs.

The proof of the theorem is similar to the Theorem 4 once we have dealt with the update of the execution period induced by the parallel composition.

Proof. Recall that α and β are behaviors of discrete component and are thus of the form $(\alpha_1 \cup \ldots \cup \alpha_n)^*$ and $(\beta_1 \cup \ldots \cup \beta_m)^*$. Without loss of generality, we can assume that $n = m = 1$, the reasoning still works for an arbitrary n and m. α_1 is of the form $?t \leqslant t_{\alpha_1} + \Delta_\alpha; \mathbf{Body}_{\alpha_1}; t_{\alpha_1} := t$.

Substituting Δ_α by $\Delta_{\alpha\beta}$ does not break the guarantee G_α. More precisely, if the sequent $\mathcal{E}, A_\alpha \vdash [(\alpha_1)^*]G_\alpha$ is valid, then the sequent $\mathcal{E}, A_\alpha \vdash [(\alpha_1{}^{\Delta_{\alpha\beta}}_{\Delta_\alpha})^*]G_\alpha$ is valid too.

The condition (d) ensures that Δ_α does not occur in the functional behavior $\mathbf{Body}_{\alpha_1}$ of α_1, *i.e.* it occurs only in the test $?t \leqslant t_{\alpha_1} + \Delta_\alpha$. Thus the substitution does not affect the functional behavior α_1, and the same proof rules used to prove the sequent $\mathcal{E}, A_\alpha \vdash [(\alpha_1)^*]G_\alpha$ can be used to prove the sequent $\mathcal{E}, A_\alpha \vdash [(\alpha_1{}^{\Delta_{\alpha\beta}}_{\Delta_\alpha})^*]G_\alpha$.

We have thus a proof of $\mathcal{E}, A_\alpha \vdash [(\alpha_1{}^{\Delta_{\alpha\beta}}_{\Delta_\alpha})^*]G_\alpha$ and of $\mathcal{E}, A_\beta \vdash [(\beta_1{}^{\Delta_{\alpha\beta}}_{\Delta_\beta})^*]G_\beta$. We apply the Theorem 4 to conclude. □

Example 47 (Proof of the composition of two controllers)**.** *We want to apply the Theorem 7 to the timed parallel composition of two controllers as in the Example 46. The conditions* (a) *to* $(c2)$ *are trivially satisfied since the two water-level controllers are independent. The condition* (d) *is also verified since neither the execution period* $\Delta_{wlCtrl1}$ *or* $\Delta_{wlCtrl2}$ *occurs in the functional behaviors of both water-level controllers. Thus the component resulting from the parallel composition of* `Controller1` *and* `Controller2` *satisfies the contract* $(A_{WlCtrl1} \wedge A_{WlCtrl2}, G_{WlCtrl1} \wedge G_{WlCtrl2})$.

Conclusion We have presented how we can compose discrete components and update their execution period during the parallel composition. We present next how we compose continuous components and still keep track of the control period, then how to compose continuous and discrete components.

4.2.3 Timed parallel composition of continuous components

When composing in parallel two continuous components, the control period of the resulting system is the minimum of respective control period. It is necessary to consider the minimum to retain safety guarantees. Indeed, if a property is guaranteed for some duration, it is also guaranteed for a shorter amount of time.

Modeling We present the textual representation of the parallel composition of two continuous components, then define the resulting behavior.

Definition 40 (Parallel composition of continuous components)**.** *The component resulting from the parallel composition of two continuous components* `A` *and* `B` *is:*

```
Compound Component: AB
Control period:
 min(ε_α, ε_β)
Inputs:
 A.inputs ∪ B.inputs
Assumptions:
 A_α ∧ A_β
Outputs:
 A.outputs ∪ B.outputs
Guarantees:
 G_α ∧ G_β
Functional behavior:
 α ⊗ β
```

The resulting inputs (resp. outputs) is the union of respective inputs (resp. outputs) as for the parallel composition of discrete components (cf Definition 38). Similarly, the resulting assumption (resp. guarantee) is the conjunction of assumptions (resp. guarantee). The control period $\varepsilon_{\alpha\beta}$ is the minimum of respective control period ε_α and ε_β. Recall that the behavior α of a continuous component is differential equation: $\dot{X} = \theta_X, \dot{t} = 1 \;\&\; H_X \wedge 0 \leqslant t \leqslant \varepsilon_\alpha$.

Definition 41 (Parallel composition of continuous components)**.** *Let α and β be two behaviors of continuous components* A *and* B *with control period ε_α and ε_β. The parallel composition $\alpha \otimes \beta$ is:*

$$\dot{X} = \theta_X, \dot{Y} = \theta_Y \;\;\&\;\; H_X \wedge H_Z \wedge 0 \leqslant t \leqslant min(\varepsilon_\alpha, \varepsilon_\beta)$$

For example, we have the parallel composition of the two water-level components. The resulting control period is the minimum of respective control period, here 0.1 seconds.

Example 48 (Composition of the two water-level)**.** *We compose the components* `Water-level1` *and* `Water-level2`*. They have respective control period $\varepsilon_{wl_1} = 0.2$ and $\varepsilon_{wl2} = 0.1$. The control period of the resulting component is thus $min(0.2, 0.15) = 0.1$ seconds. The resulting continuous component is:*

$$\dot{wl_1} = fin - fout_1, \dot{wl_2} = fout_1 - fout_2, \dot{t} = 1 \;\;\&\;\; wl_1 \geqslant 0 \wedge wl_2 \geqslant 0 \wedge 0 \leqslant t \leqslant 0.15$$

Algebraic properties As for the timed parallel composition of discrete components, we retain commutativity and associativity for the timed parallel composition of continuous component.

Proposition 6 (Commutativity and Associativity)**.** *Let α, β and γ be behaviors of continuous components* A*,* B *and* C*, and of respective control period ε_α, ε_β and ε_γ.*

$$\begin{array}{lcll} \alpha \otimes \beta & = & \beta \otimes \alpha & \textit{(Commutativity)} \\ (\alpha \otimes \beta) \otimes \gamma & = & \alpha \otimes (\beta \otimes \gamma) & \textit{(Associativity)} \end{array}$$

Proof. Recall that the behavior α of a continuous component is the differential equation: $\dot{X} = \theta_X, \dot{t} = 1 \;\&\; H_X \wedge 0 \leqslant t \leqslant \varepsilon_\alpha$. Idem for β of the form $\dot{Y} = \theta_Y, \dot{t} = 1 \;\&\; H_Y \wedge 0 \leqslant t \leqslant \varepsilon_\beta$ and γ of the form $\dot{Z} = \theta_Z, \dot{t} = 1 \;\&\; H_Z \wedge 0 \leqslant t \leqslant \varepsilon_\gamma$.

We first prove the commutativity property. The basic idea is that the operator $min(.,.)$ is commutative and associative.

We unfold the left-hand side and show it is equivalent to the right-hand side.

$$\begin{array}{rcll} \alpha \otimes \beta & \triangleq & \dot{X} = \theta_X, \dot{Y} = \theta_Y \;\;\&\;\; H \wedge 0 \leqslant t \leqslant min(\varepsilon_\alpha, \varepsilon_\beta) & (i) \\ & \triangleq & \dot{Y} = \theta_Y, \dot{X} = \theta_X \;\;\&\;\; H \wedge 0 \leqslant t \leqslant min(\varepsilon_\alpha, \varepsilon_\beta) & (ii) \\ & \triangleq & \dot{Y} = \theta_Y, \dot{X} = \theta_X \;\;\&\;\; H \wedge 0 \leqslant t \leqslant min(\varepsilon_\beta, \varepsilon_\alpha) & (iii) \\ & \triangleq & \beta \otimes \alpha & \end{array}$$

The step (i) consists of the unfolding of the definition of the parallel composition operator $\otimes$ for continuous behaviors. The second step (ii) is the reordering of variables in the ODE. The step (iii) replace $min(\varepsilon_\alpha, \varepsilon_\beta)$ by $min(\varepsilon_\beta, \varepsilon_\alpha)$ since they are equal. Finally, we fold the definition to obtain $\beta \otimes \alpha$.

For the associativity, we unfold the definition and use the associativity of the $min(.,.)$ operator to show that $(\alpha \otimes \beta) \otimes \gamma = \alpha \otimes (\beta \otimes \gamma)$.

$$\begin{array}{lr}
(\alpha \otimes \beta) \otimes \gamma & \\
\triangleq \left(\dot{X} = \theta_X, \dot{Y} = \theta_Y \;\&\; H \wedge ControlPeriod_{\alpha\beta}\right) \otimes \gamma & \text{(Unfolding)} \\
\triangleq \dot{X} = \theta_X, \dot{Y} = \theta_Y, \dot{Z} = \theta_Z \;\&\; H \wedge ControlPeriod_{(\alpha\beta)\gamma} & \text{(Unfolding)} \\
\triangleq \dot{X} = \theta_X, \dot{Y} = \theta_Y, \dot{Z} = \theta_Z \;\&\; H \wedge ControlPeriod_{\alpha(\beta\gamma)} & (i) \\
\triangleq \alpha \otimes \left(\dot{Y} = \theta_Y, \dot{Z} = \theta_Z \;\&\; H \wedge ControlPeriod_{\beta\gamma}\right) & \text{(Folding)} \\
\triangleq \alpha \otimes (\beta \otimes \gamma) & \text{(Folding)}
\end{array}$$

The step (i) use the equality $min(min(\varepsilon_\alpha, \varepsilon_\beta), \varepsilon_\gamma) = min(\varepsilon_\alpha, min(\varepsilon_\beta, \varepsilon_\gamma))$. We deduce that

$$\begin{array}{rl}
& ControlPeriod_{(\alpha(\beta\gamma)} \\
= & 0 \leqslant t \leqslant min(min(\varepsilon_\alpha, \varepsilon_\beta), \varepsilon_\gamma) \\
= & 0 \leqslant t \leqslant min(\varepsilon_\alpha, min(\varepsilon_\beta, \varepsilon_\gamma)) \\
= & ControlPeriod_{\alpha(\beta\gamma)}
\end{array}$$

□

Modular proof As for the timed parallel composition of discrete components, we retain contracts through the timed parallel composition of continuous components.

Theorem 8. *Let α and β be two behaviors of continuous components* `A` *and* `B` *with respective contracts (A_α, G_α) and (A_β, G_β). Assume that we have a proof tree of $\mathcal{E}, A_\alpha \vdash [\alpha]G_\alpha$ and $\mathcal{E}, A_\beta \vdash [\beta]G_\beta$. Then, under the conditions:*

- (a) $BV(\alpha) \cap BV(\beta) = \varnothing$,
- (b_1) $BV(\alpha) \cap FV(G_\beta) = \varnothing$ *and* $BV(\alpha) \cap FV(H_Y) = \varnothing$
- (b_2) $BV(\beta) \cap FV(G_\alpha) = \varnothing$ *and* $BV(\beta) \cap FV(H_X) = \varnothing$,
- (c_1) $A_\alpha \vdash \forall^\beta(G_\beta \to A_\alpha)$,
- (c_2) $A_\beta \vdash \forall^\alpha(G_\alpha \to A_\beta)$.

We automatically obtain a proof tree of the sequent:

$$\mathcal{E}, A_\alpha, A_\beta \vdash [\alpha \otimes \beta](G_\alpha \wedge G_\beta)$$

We still have the restriction on separated outputs and that the guarantees of a component must not refer to the outputs of an other component. We also require that the guarantee of a component does not break the assumption under which the second component operates.

Proof. $\alpha \otimes \beta$ is the same as $\alpha' \circ \beta'$ where $\alpha' \triangleq \dot{X} = \theta_X \;\&\; H_X \wedge ControlPeriod_{\alpha\beta}$ and $\beta' \triangleq \dot{Y} = \theta_Y \;\&\; H_Y \wedge ControlPeriod_{\alpha\beta}$. $\circ_c$ is the parallel continuous composition in Definition 21. If we have a proof of $\mathcal{E}, A_\alpha \vdash [\alpha']G_\alpha$ and $\mathcal{E}, A_\beta \vdash [\beta']G_\beta$, we can conclude by the Theorem 1.

We show that if we have a proof tree Π_α of $\mathcal{E}, A_\alpha \vdash [\alpha]G_\alpha$, then we have a proof tree of $\mathcal{E}, A_\alpha \vdash [\alpha']G_\alpha$.

If $min(\varepsilon_\alpha, \varepsilon_\beta) = \varepsilon_\alpha$, then $\alpha' = \alpha$ and we have nothing to prove.

If $min(\varepsilon_\alpha, \varepsilon_\beta) = \varepsilon_\beta$, we have $\alpha' \triangleq \dot{X} = \theta_X \ \ \& \ \ H_X \wedge 0 \leqslant t \leqslant \varepsilon_\beta$. By assumption, we have a proof of the sequent $\mathcal{E}, A_\alpha \vdash [\dot{X} = \theta_X \ \ \& \ \ H_X \wedge 0 \leqslant t \leqslant \varepsilon_\alpha]G_\alpha$. Since $\varepsilon_\alpha \geqslant \varepsilon_\beta$, α' considers the evolution of the differential equation during a shorter amount of time, we can then deduce the validity of the sequent $\mathcal{E}, A_\alpha \vdash [\alpha']G_\alpha$.

□

Example 49 (Composition of two water-level)**.** *We want to apply the Theorem 8 to the timed parallel composition of two water-level as in the Example 48. The conditions* (a) *to* (c2) *are trivially satisfied since the two water-level are independent. Thus the component resulting from the parallel composition of* `Water-level1` *and* `Water-level2` *satisfies the contract* $(A_{Wl1} \wedge A_{Wl2}, G_{Wl1} \wedge G_{Wl2})$*.*

Conclusion We have presented how to integrate timing aspects in our component-based approach for purely discrete systems and purely continuous systems. But the main interest of hybrid systems is when the systems are hybrid, *i.e.* when there is an interaction between continuous components and discrete components. It is also during this phase that a lot of problems can arise and that our approach help to solve.

4.2.4 Timed parallel composition of a discrete and a continuous component

In this section, we present the composition of a discrete and a continuous component. It is almost the same as the presentation in Section 4.1, but a more general integration of the notion of control period and execution periods to pass to the composition of different hybrid systems. We require that the control period of the continuous component is smaller to the execution period of the discrete component to be able to retain contracts through composition.

Modeling We present the textual representation of the parallel composition of a discrete component with a continuous component, then define the resulting behavior in Definition 39.

Definition 42 (Parallel composition of a discrete and a continuous component)**.** *The component resulting from the parallel composition of a discrete component* `A` *and a continuous component* `B` *is defined by:*

`Compound Component: AB`
`Control period:`
 ε_β
`Period:`
 Δ_α
`Inputs:`
 `A.inputs` $\cup$ `B.inputs`
`Assumptions:`
 $A_\alpha \ \wedge \ A_\beta$
`Outputs:`
 `A.outputs` $\cup$ `B.outputs`
`Guarantees:`
 $G_\alpha \ \wedge \ G_\beta$
`Functional behavior:`
 $\alpha \ \otimes \ \beta$

The control period and the execution period do not change. The behavior of the resulting component is defined using these two values. A discrete component may be the result of the parallel composition of several discrete sub-components and the behavior α of a component `A` is of the following form: $(\alpha_1 \cup \ldots \cup \alpha_n)^*$, where α_i represent the behavior of sub-components. Each of them have a notification variable t_{α_i} associated.

Definition 43 (Discrete-Continuous)**.** *Let α be the behavior of a discrete component* `A` *with execution period Δ_α and β be the behavior of a continuous component* `B` *with control period ε_β. The parallel composition $\alpha \otimes \beta$ is:*

$$\left(\alpha_1 \cup \ldots \cup \alpha_n \cup \dot{X} = \theta \ \ \& \ \ H \wedge ControlPeriod_{\alpha\beta} \right)^*$$

where $ControlPeriod_{\alpha\beta} = \bigwedge_{1 \leqslant i \leqslant n} 0 \leqslant t - t_{\alpha_i} \leqslant \Delta_\alpha$.

The control period formula $ControlPeriod_{\alpha\beta}$ is the conjunction of control period formulas for each sub-component behavior α_i of α. It is necessary to retain the associativity as shown in the next Subsection 4.2.5. A simple example of the parallel composition of a discrete component and a continuous components is the water-tank as in the Example 38.

Example 50 (Water-tank)**.** *The component* `Water-tank1` *is obtained by the composition of components* `Controller1` *and* `Water-level1`*. It is similar the behavior in Example 38.*

Example 51 (Water-plant)**.** *In Example 46, we have composed the two water-level controllers. Let denote the resulting component by* `Controller12`*. In Example 48, we have composed the two water-level. Let denote the resulting component by* `Water-level12`*. By composing* `Controller12` *with* `Water-level12`*, we obtain the water-plant component* `Water-plant`*. The execution period Δ_{wp} is the execution period of the parallel composition of both controllers, i.e. $\Delta_{wlCtrl1} + \Delta_{wlCtrl2} = 0.03 + 0.02 = 0.05$. The control period ε_{wp} is the control period of the parallel composition of the two water-level, i.e. $min(\varepsilon_{wl1}, \varepsilon_{wl2} = min(0.2, 0.1) = 0.1$.*

Modular proof As for the two previous case, we have a theorem ensuring that we retain contracts through composition. It is in fact almost the same as Theorem 6. The difference is that we explicitly distinguish the notion of *control period* and of *execution period.* Plus, we are not restricted to a system with one controller and one plant, but possibly several controllers already in parallel with several plants already in parallel as in Example 51.

Theorem 9. *Let α and β be two behaviors of a discrete component* `A` *and a continuous component* `B` *with respective contracts (A_α, G_α) and (A_β, G_β), control period ε_β and execution period Δ_α. Assume that we have a proof tree of $\mathcal{E}, A_\alpha \vdash [\alpha]G_\alpha$ and $\mathcal{E}, A_\beta \vdash [\beta]G_\beta$. Then, under the conditions:*

(*a*) $BV(\alpha) \cap BV(\beta) = \varnothing$,

(*b*$_1$) $BV(\alpha) \cap FV(G_\beta) = \varnothing$ *and* $BV(\alpha) \cap FV(H_\beta)$,

(*b*$_2$) $BV(\beta) \cap FV(G_\alpha) = \varnothing$,

(*c*$_1$) $A_\alpha \vdash \forall^\beta (G_\beta \rightarrow A_\alpha)$,

(c_2) $A_\beta \vdash \forall^\alpha(G_\alpha \rightarrow A_\beta)$,

(d) $\varepsilon_\beta \geqslant \Delta_\alpha$.

We automatically obtain a proof tree of the sequent:

$$\mathcal{E}, A_\alpha, A_\beta \vdash [\alpha \otimes \beta](G_\alpha \wedge G_\beta)$$

Proof. The proof is exactly the same as for Theorem 6. □

Example 52 (Water-plant)**.** *The parallel composition of components* `Controller12` *and* `Water-level12` *satisfies the conditions* (a) *to* (c_2) *of the Theorem 9 as presented in Subsection 3.5.5. Plus, we have seen in the Example 51 that the execution period* Δ_{wp} *is* 0.05 *and the control period* ε_{wp} *is* 0.1*. We have* $\varepsilon_{wp} \geqslant \Delta_{wp}$*; the condition* ($d$) *is thus respected. By application of the Theorem 9, the component* `Water-plant` *satisfies the contract* $(A_{wlCtrl1} \wedge A_{wlCtrl2} \wedge A_{wl1} \wedge A_{wl2}, G_{wlCtrl1} \wedge G_{wlCtrl2} \wedge G_{wl1} \wedge G_{wl2})$*.*

4.2.5 Timed parallel composition of two general components

A general component can be discrete and continuous, *e.g.* a CCS. The behavior α of such component has the canonical form $(\mathbf{disc}_\alpha \cup \mathbf{cont}_\alpha)^*$. If we want to add a discrete component, *e.g.* a program monitoring an other plant but which have to run on this CPU, in parallel, then we have to be careful that the new duration of execution of the resulting discrete subcomponent does not exceed the control period of the plant.

Modeling We present the definition of timed parallel composition for general components. It aggregates the previous definitions.

Definition 44 (Parallel composition of two general components)**.** *The component* `AB` *resulting from the parallel composition of a general component* `A` *and a general component* `B` *is defined by:*

```
Compound Component: AB
Control period:
 min(ε_α, ε_β)
Period:
 Δ_α + Δ_β
Inputs:
 A.inputs ∪ B.inputs
Assumptions:
 A_α ∧ A_β
Outputs:
 A.outputs ∪ B.outputs
Guarantees:
 G_α ∧ G_β
Functional behavior:
 α ⊗ β
```

The resulting control period is the minimum of respective control period as in Definition 40. The resulting execution period is the sum of respective execution periods as in Definition 39.

To define the parallel composition $\alpha \otimes \beta$ of the behaviors α and β, we define the formula $ControlPeriod_{\alpha\beta}$ according to every possible association of α and β being discrete, continuous or general behaviors. Recall that the the canonical form of the discrete part of α (resp. β) is $(\alpha_1 \cup \ldots \alpha_n)$ (resp. $(\beta_1 \cup \ldots \beta_n)$).

Definition 45 ($ControlPeriod_{\alpha\beta}$)**.** *Let α and β be behaviors of components* `A` *and* `B`*. Their respective execution periods are defined by Δ_α and Δ_β and their respective control period by ε_α and ε_β. The formula $ControlPeriod_{\alpha\beta}$ is defined by:*

$$\begin{cases} 0 \leqslant t \leqslant min(\varepsilon_\alpha, \varepsilon_\beta) & \textit{if } \alpha \textit{ and } \beta \textit{ are continuous} \\ \bigwedge\limits_{1 \leqslant j \leqslant m} 0 \leqslant t - t_{\beta_j} \leqslant \Delta_\beta & \textit{if } \alpha \textit{ is continuous and } \beta \textit{ is discrete} \\ \bigwedge\limits_{1 \leqslant i \leqslant n} 0 \leqslant t - t_{\alpha_i} \leqslant \Delta_\alpha & \textit{if } \alpha \textit{ is discrete and } \beta \textit{ is continuous} \\ \bigwedge\limits_{1 \leqslant j \leqslant m} (0 \leqslant t - t_{\beta_j} \leqslant \Delta_\alpha + \Delta_\beta) \wedge (ControlPeriod_\alpha)_{\Delta_\alpha}^{\Delta_\alpha + \Delta_\beta} & \textit{if } \alpha \textit{ is general and } \beta \textit{ is discrete} \\ \bigwedge\limits_{1 \leqslant i \leqslant n} (0 \leqslant t - t_{\alpha_i} \leqslant \Delta_\alpha + \Delta_\beta) \wedge (ControlPeriod_\beta)_{\Delta_\beta}^{\Delta_\alpha + \Delta_\beta} & \textit{if } \alpha \textit{ is discrete and } \beta \textit{ is general} \\ (ControlPeriod_\alpha)_{\Delta_\alpha}^{\Delta_\alpha + \Delta_\beta} \wedge (ControlPeriod_\beta)_{\Delta_\alpha}^{\Delta_\alpha + \Delta_\beta} & \textit{if } \alpha \textit{ and } \beta \textit{ are general} \end{cases}$$

The first case where α and β are both continuous is presented in Subsection 4.2.3. The second and third case, where one is discrete and the other continuous, is presented in Subsection 4.2.2. The fourth and fifth case is the conjunction of the two following formulas, $\bigwedge\limits_{1 \leqslant j \leqslant m} (0 \leqslant t - t_{\beta_j} \leqslant \Delta_\alpha + \Delta_\beta)$ and $(ControlPeriod_\alpha)_{\Delta_\alpha}^{\Delta_\alpha + \Delta_\beta}$. The resulting execution period is the sum of each respective period since both α and β exhibit discrete behaviors. The left conjunct is obtained as in the second case, but with the updated value for the execution period. The right conjunct is the control period formula of α already obtained by parallel composition with also an updated value for the execution period. The control period formula in the last case, when both α and β are general, is just the conjunction of both control period formulas with an update on the execution period.

Having defined the formula $ControlPeriod_{\alpha\beta}$ for every possible associations, we define the parallel composition for general components.

Definition 46 (Parallel composition of general behaviors)**.** *Let α and β be behaviors of general components. Remember that they respect the canonical form $\alpha \triangleq \big(\boldsymbol{disc}_\alpha \cup (\dot{X} = \Theta_X \ \& \ H_X \wedge ControlPeriod_\alpha)\big)^*$ and $\beta \triangleq \big(\boldsymbol{disc}_\beta \cup (\dot{Y} = \Theta_Y \ \& \ H_Y \wedge ControlPeriod_\beta)\big)^*$. The parallel composition $\alpha \otimes \beta$ is:*

$$\Big(\ \boldsymbol{disc}_\alpha{}_{\Delta_\alpha}^{\Delta_{\alpha\beta}} \cup \boldsymbol{disc}_\beta{}_{\Delta_\beta}^{\Delta_{\alpha\beta}} \cup (\dot{X} = \theta_X, \dot{Y} = \theta_Y \ \& \ H_X \wedge H_Y \wedge ControlPeriod_{\alpha\beta}) \ \Big)^*$$

Example 53 (Water-Plant)**.** *In Example 51, we obtain the water-plant example by first composing the two controllers and the two water-level separately and then composing each resulting component. Here, we assume that we have composed the water-level of the first tank with its controller, the water-level of the second tank with its controller, resulting in two general components representing the first tank and the second tank. We can now compose them to obtain the water-plant system.*

The two ways to obtain the water-plant system are equivalent thanks to the commutativity and associativity properties that we retain.

Algebraic properties We retain the commutativity and associativity properties necessary to a modular component-based approach.

Proposition 7 (Commutativity and associativity). *Let α, β and γ be behaviors of general components* `A`*,* `B` *and* `C`*, of respective execution periods Δ_α, Δ_β and Δ_γ, and of respective control period ε_α, ε_β and ε_γ.*

$$\begin{array}{lcll} \alpha \otimes \beta & = & \beta \otimes \alpha & \textit{(Commutativity)} \\ (\alpha \otimes \beta) \otimes \gamma & = & \alpha \otimes (\beta \otimes \gamma) & \textit{(Associativity)} \end{array}$$

Proof. Recall that the behaviors α, β and γ are of the respective canonical form $\big(\mathbf{disc}_\alpha \cup (\dot{X} = \Theta_X \ \ \& \ \ H_X \wedge \mathit{ControlPeriod}_\alpha)\big)^*$, $\big(\mathbf{disc}_\beta \cup (\dot{Y} = \Theta_Y \ \ \& \ \ H_Y \wedge \mathit{ControlPeriod}_\beta)\big)^*$ and $\big(\mathbf{disc}_\gamma \cup (\dot{Z} = \Theta_Z \ \ \& \ \ H_Z \wedge \mathit{ControlPeriod}_\gamma)\big)^*$.

The proof of commutativity is obtained by first unfolding the definition (step (i)). We use the commutativity of the $\cup$ operator to reorder the discrete parts (step (ii)). We reorder the variables in the differential equation and use the commutativity of the conjunction $\wedge$ to pass from $H_X \wedge H_Y$ to $H_Y \wedge H_X$ (step (iii)). The fourth step (iv) is justified by the equalities $\mathit{ControlPeriod}_{\alpha\beta} = \mathit{ControlPeriod}_{\beta\alpha}$ and $\Delta_{\alpha\beta} = \Delta_{\beta\alpha}$. We fold the definition to obtain $\beta \otimes \alpha$ (step (v)).

$$\begin{array}{ll} \alpha \otimes \beta & \\ = \big((\mathbf{disc}_\alpha)_{\Delta_\alpha}^{\Delta_{\alpha\beta}} \cup (\mathbf{disc}_\beta)_{\Delta_\beta}^{\Delta_{\alpha\beta}} \cup \dot{X} = \theta_X, \dot{Y} = \theta_Y \ \ \& \ \ H_X \wedge H_Y \wedge \mathit{ControlPeriod}_{\alpha\beta}\big)^* & (i) \\ = \big((\mathbf{disc}_\beta)_{\Delta_\beta}^{\Delta_{\alpha\beta}} \cup (\mathbf{disc}_\alpha)_{\Delta_\alpha}^{\Delta_{\alpha\beta}} \cup \dot{X} = \theta_X, \dot{Y} = \theta_Y \ \ \& \ \ H_X \wedge H_Y \wedge \mathit{ControlPeriod}_{\alpha\beta}\big)^* & (ii) \\ = \big((\mathbf{disc}_\beta)_{\Delta_\beta}^{\Delta_{\alpha\beta}} \cup (\mathbf{disc}_\alpha)_{\Delta_\alpha}^{\Delta_{\alpha\beta}} \cup \dot{Y} = \theta_Y, \dot{X} = \theta_X \ \ \& \ \ H_Y \wedge H_X \wedge \mathit{ControlPeriod}_{\alpha\beta}\big)^* & (iii) \\ = \big((\mathbf{disc}_\beta)_{\Delta_\beta}^{\Delta_{\beta\alpha}} \cup (\mathbf{disc}_\alpha)_{\Delta_\alpha}^{\Delta_{\beta\alpha}} \cup \dot{Y} = \theta_Y, \dot{X} = \theta_X \ \ \& \ \ H_Y \wedge H_X \wedge \mathit{ControlPeriod}_{\beta\alpha}\big)^* & (iv) \\ = \beta \otimes \alpha & (v) \end{array}$$

To prove the associativity property, we first unfold the definition of the parallel composition (steps (i) and (ii)). We use the equalities $\mathit{ControlPeriod}_{(\alpha\beta)\gamma} = \mathit{ControlPeriod}_{\alpha(\beta\gamma)}$ and $\Delta_{(\alpha\beta)\gamma} = \Delta_{\alpha(\beta\gamma)}$ in the step (iii). We fold the definition in the steps (iv) and (v).

$$\begin{array}{rll} & (\alpha \otimes \beta) \otimes \gamma & \\ = & \Big(\big((\mathbf{disc}_\alpha)_{\Delta_\alpha}^{\Delta_{\alpha\beta}} \cup (\mathbf{disc}_\beta)_{\Delta_\beta}^{\Delta_{\alpha\beta}} & \\ & \cup \dot{X} = \theta_X, \dot{Y} = \theta_Y \ \ \& \ \ H_X \wedge H_Y \wedge \mathit{ControlPeriod}_{\alpha\beta}\big)^*\Big) \otimes \gamma & (i) \\ = & \big((\mathbf{disc}_\alpha)_{\Delta_\alpha}^{\Delta_{(\alpha\beta)\gamma}} \cup (\mathbf{disc}_\beta)_{\Delta_\beta}^{\Delta_{(\alpha\beta)\gamma}} \cup (\mathbf{disc}_\gamma)_{\Delta_\gamma}^{\Delta_{(\alpha\beta)\gamma}} & \\ & \cup \ \dot{X} = \theta_X, \dot{Y} = \theta_Y, Z = \theta_Z \ \ \& \ \ H_X \wedge H_Y \wedge H_Z \wedge \mathit{ControlPeriod}_{(\alpha\beta)\gamma}\big)^* & (ii) \\ = & \big((\mathbf{disc}_\alpha)_{\Delta_\alpha}^{\Delta_{\alpha(\beta\gamma)}} \cup (\mathbf{disc}_\beta)_{\Delta_\beta}^{\Delta_{\alpha(\beta\gamma)}} \cup (\mathbf{disc}_\gamma)_{\Delta_\gamma}^{\Delta_{\alpha(\beta\gamma)}} & \\ & \cup \ \dot{X} = \theta_X, \dot{Y} = \theta_Y, Z = \theta_Z \ \ \& \ \ H_X \wedge H_Y \wedge H_Z \wedge \mathit{ControlPeriod}_{\alpha(\beta\gamma)}\big)^* & (iii) \\ = & \alpha \otimes \Big(\big((\mathbf{disc}_\beta)_{\Delta_\beta}^{\Delta_{\beta\gamma}} \cup (\mathbf{disc}_\gamma)_{\Delta_\gamma}^{\Delta_{\beta\gamma}} & \\ & \quad \cup \ \dot{Y} = \theta_Y, Z = \theta_Z \ \ \& \ \ H_Y \wedge H_Z \wedge \mathit{ControlPeriod}_{\beta\gamma}\big)^*\Big) & (iv) \\ = & \alpha \otimes (\beta \otimes \gamma) & (v) \end{array}$$

□

Modular proof We retain also the respective contracts through the timed parallel composition.

Theorem 10. *Let α and β be two behaviors of general components* `A` *and* `B` *with respective contracts (A_α, G_α) and (A_β, G_β), control period ε_α and ε_β, and execution periods Δ_α and Δ_β. Assume that we have a proof tree of $\mathcal{E}, A_\alpha \vdash [\alpha]G_\alpha$ and $\mathcal{E}, A_\beta \vdash [\beta]G_\beta$. Then, under the conditions:*

- (a) $BV(\alpha) \cap BV(\beta) = \varnothing$,
- (b_1) $BV(\alpha) \cap FV(G_\beta) = \varnothing$,
- (b_2) $BV(\beta) \cap FV(G_\alpha) = \varnothing$,
- (c_1) $A_\alpha \vdash \forall^\beta(G_\beta \rightarrow A_\alpha)$,
- (c_2) $A_\beta \vdash \forall^\alpha(G_\alpha \rightarrow A_\beta)$,
- (d) *The period Δ_α (resp. Δ_β) does not occur in the functional behavior of* `A` *or G_α (resp. functional behavior of* `A` *or G_β),*
- (e) $min(\varepsilon_\alpha, \varepsilon_\beta) \geqslant \Delta_\alpha + \Delta_\beta$.

We obtain automatically a proof tree of the sequent:

$$\mathcal{E}, A_\alpha, A_\beta \vdash [\alpha \otimes \beta](G_\alpha \wedge G_\beta)$$

The conditions are the aggregations of the conditions of Theorem 7 and Theorem 8. The proof of this theorem is achieved by applying the three theorems of the previous subsections.

Proof. Remember that α and β are behaviors of general components. They respect the canonical form, and thus we have $\alpha \triangleq \big(\mathbf{disc}_\alpha \cup (\dot{X} = \Theta_X \ \& \ H_X \wedge ControlPeriod_\alpha)\big)^*$ and $\beta \triangleq \big(\mathbf{disc}_\beta \cup (\dot{Y} = \Theta_Y \ \& \ H_Y \wedge ControlPeriod_\beta)\big)^*$. We want to prove the validity of the sequent $\mathcal{E}, A_\alpha, A_\beta \vdash [\big((\mathbf{disc}_\alpha)_{\Delta_\alpha}^{\Delta_{\alpha\beta}} \cup (\mathbf{disc}_\beta)_{\Delta_\beta}^{\Delta_{\alpha\beta}} \cup (\dot{X} = \theta_X, \dot{Y} = \theta_Y \ \& \ H_X \wedge H_Y \wedge ControlPeriod_{\alpha\beta})\big)^*](G_\alpha \wedge G_\beta)$.

The behavior resulting from the parallel composition is given by $\big((\mathbf{disc}_\alpha)_{\Delta_\alpha}^{\Delta_{\alpha\beta}} \cup (\mathbf{disc}_\beta)_{\Delta_\beta}^{\Delta_{\alpha\beta}} \cup (\dot{X} = \theta_X, \dot{Y} = \theta_Y \ \& \ H_X \wedge H_Y \wedge ControlPeriod_{\alpha\beta})\big)^*$ and it can be expressed as the successive composition of discrete parts on one side, the continuous part on the other side, and then the composition of both sides. More precisely, it is equivalent to

$$\big((\mathbf{disc}_\alpha)^* \otimes (\mathbf{disc}_\beta)^*\big) \otimes \big((\dot{X} = \theta_X \ \& \ H_X \wedge ControlPeriod_\alpha) \otimes (\dot{Y} = \theta_Y \ \& \ H_Y \wedge ControlPeriod_\beta)\big)$$

We successively apply the Theorems 7, 8 and 9 to conclude. □

Example 54 (Water-plant)**.** *We already a proof that the* `water-tank1` *component satisfies its contract (A_{wt1}, G_{wt1}) and that the* `water-tank2` *component satisfies its contract (A_{wt2}, G_{wt2}). The conditions of the Theorem 10 are verified and we have that the resulting component* `Water-plant` *satisfies the conjunction of contracts $(A_{wt1} \wedge A_{wt2}, G_{wt1} \wedge G_{wt2})$.*

Conclusion

In this section, we have presented how to extend our previous component-based approach to take into account the timing constraints inherent to the design of a Computer-Controlled System. We have proved that we retain the commutativity and associativity properties, essential to scale up to realistic systems. Finally, we state and prove theorems to retain contracts through the parallel conditions. Theses results give us confidence in the ability of our approach to be adapted to new challenges that will arise when confronted to realistic industrial systems.

The next two sections present two adaptions of our parallel composition operator to handle modes and causal composition. It features composition theorems with relaxed condition to retain contracts.

4.3 Handling of modes

Complex computer-controlled systems often exhibits several *modes*. For example, in a water-plant, there is the *nominal* mode where the treatment of water is working normally and the water-level in tanks should be between a predefined range. And there is the *error* mode for when there is an anomaly, *e.g.* a worker push an emergency button, and we just want to ensure that there is no overflow by closing all the inlet faucets. It allows to model systems in a degraded environment and still ensure some important safety properties.

The obvious approach will be to model each mode as a component behavior, and then apply our parallel composition operator to obtain the global system. But by design, two distinct modes have the same outputs and it is not possible to apply the Theorem 4 since the condition (a) (the outputs must be separated) is not satisfied. The trick is that the two modes do not execute simultaneously. They are exclusive; we cannot be in the two cases at the same time. We exploit this exclusion to slightly alter our definition of component in Subsection 3.1.1 similarly to our work in the previous Sections 4.1 and 4.2. We prove a similar result to Theorem 4 while relaxing the condition of the disjoints outputs for this design.

Here, we present our approach only with discrete components. To our knowledge, the definition of modes is for discrete systems since it is a design choice performed by an engineer. The change of mode of a plant is enforced by the actuation of a controller.

4.3.1 Modular modeling

The designer provides an event E under the form of a formula which characterizes a mode, *e.g.* $Error = 0$. Each event is added in the guard as for the time test in Section 4.2. The events can change from one to the other at every execution cycle.

We define a component `A` as in Subsection 3.1.1 to which we add a event E_α. From this specification, we derive a behavior as a hybrid program which models the desired working.

Component: A
Event:
 E_α
Inputs:
 A.inputs $\cup$ B.inputs
Assumptions:

$A_\alpha \;\wedge\; A_\beta$

`Outputs:`

`A.outputs` $\cup$ `B.outputs`

`Guarantees:`

$G_\alpha \;\wedge\; G_\beta$

`Functional behavior:`

α

We recall that α is the functional behavior of a discrete component and may result from multiple previous compositions. It is thus of the form $\alpha \triangleq \alpha_1 \cup \ldots \cup \alpha_n$ where α_i are behaviors of functional sub-components.

Definition 47 (Mode)**.** *Let α be the functional behavior of a discrete component* `A` *with event E_α. The canonical structure of a* mode *is given by the hybrid program:*

$$\Big(\bigcup_{1 \leqslant i \leqslant n} ?E_\alpha ; \alpha_i \Big)^*$$

We distribute the event formula E_α to every sub-components α_i of the component α.

We complicate the water-tank example by adding to the nominal mode, (cf Subsection 3.5.3), an emergency mode. The event formula is $error = 0$ for the nominal mode and $error = 1$ for the error mode. We introduce the variable $error$ representing the pushing of an emergency button. It is assumed to be an external variable; we add in the environment the fact that $error = 1 \vee error = 0$.

Example 55 (Error mode)**.** *The functional behavior of the error mode is $fin := 0; fout := 0$. We close both the inlet and outlet faucet and the system does not evolve anymore. It implies that the variable $fout$ is no longer part of the environment, but an output of the component* `Error`*. The behavior is*

$$(?error = 1; fin := 0; fout := 0)^*$$

The composition of two modes is obtained by applying our parallel composition operator as in Definition 22.

Definition 48 (Mode composition)**.** *Let* `A` *and* `B` *be two discrete components with respective behavior $\alpha \triangleq \big(\bigcup_{1 \leqslant i \leqslant n} ?E_\alpha ; \alpha_i\big)^*$ and $\beta \triangleq \big(\bigcup_{1 \leqslant j \leqslant m} ?E_\beta ; \beta_j\big)^*$, and respective events E_α and E_β. The behavior of the composition of* `A` *and* `B` *is defined as:*

$$\alpha \circ \beta \triangleq \Big(\bigcup_{1 \leqslant i \leqslant n} ?E_\alpha ; \alpha_i \cup \bigcup_{1 \leqslant j \leqslant m} ?E_\beta ; \beta_j \Big)^*$$

We use the parallel composition operator of Definition 22. We do not explicitly require that the events should be exclusive for the composition. We can model systems where two modes can overlap, but we will not be able to retain contracts through the parallel composition.

Example 56 (Composition of component `Controller` and component `Error`)**.** *From Definition 30, remember that the functional behavior of the water-level controller component,* `Controller`*, is:*

$$wlm := wl; (?wlm \geqslant 6.5; fin := 0) \cup (?wlm \leqslant 3.5; fin := 1);$$

We drop the timing constraint for the sake of clarity, but they can be added without problem. By applying the Definition 48, the behavior of the resulting component is

$$\left(\begin{array}{l} (?error = 1; fin := 0; fout := 0) \\ \cup\big(?error = 0; wlm := wl; (?wlm \geqslant 6.5; fin := 0) \cup (?wlm \leqslant 3.5; fin := 1)\big) \end{array}\right)^*$$

4.3.2 Modular proof

As in Section 3.3, we associate contracts to each component and prove that each component satisfies it. To obtain automatically the satisfaction of the conjunction of contracts by the global system, we have the following theorem.

Theorem 11. *Let α and β be two behaviors of discrete components* `A` *and* `B` *with respective contracts (A_α, G_α) and (A_β, G_β), mode events E_α and E_β. Assume that we have a proof tree of $\mathcal{E}, A_\alpha \vdash [\alpha]G_\alpha$ and $\mathcal{E}, A_\beta \vdash [\beta]G_\beta$. Then, under the conditions:*

- (a) $(E_\alpha \wedge E_\beta) \rightarrow \bot$,
- (b_1) $FV(G_\alpha) \cap \Big(BV(\beta)\backslash\big(BV(\alpha) \cap BV(\beta)\big)\Big) = \varnothing$,
- (b_2) $FV(G_\beta) \cap \Big(BV(\alpha)\backslash\big(BV(\beta) \cap BV(\alpha)\big)\Big) = \varnothing$,
- (c_1) $A_\alpha \vdash \forall^\beta(G_\beta \rightarrow A_\alpha)$,
- (c_2) $A_\beta \vdash \forall^\alpha(G_\alpha \rightarrow A_\beta)$.

We have a proof of:

$$\mathcal{E}, A_\alpha, A_\beta \vdash [\alpha \otimes \beta](E_\alpha \rightarrow G_\alpha \wedge E_\beta \rightarrow G_\beta)$$

We relax the condition that the outputs must be separated, *i.e.* that $BV(\alpha) \cap BV(\beta) = \varnothing$. To still guarantee the soundness of the composition, we add the requirement that the two events are exclusive, *i.e.* $(E_\alpha \wedge E_\beta) \rightarrow \bot$. Since the we relax the condition on separated outputs, we have to be more careful for the statement of conditions $(b1)$ and $(b2)$. Indeed, the guarantee G_α of a component may refer to a common output of both components, *e.g.* fin in our water-tank example. So, we forbid G_α to refer to outputs of `B` that are not also outputs of `A`, hence the condition$FV(G_\alpha) \cap \Big(BV(\beta)\backslash\big(BV(\alpha) \cap BV(\beta)\big)\Big) = \varnothing$.

Proof. Let α and β be two behaviors. They respect the canonical form $\big(\bigcup_{1 \leqslant i \leqslant n} ?E_\alpha; \alpha_i\big)^*$ and $\big(\bigcup_{1 \leqslant j \leqslant m} ?E_\beta; \beta_j\big)^*$. To simplify the proof, we assume that n and m are reduced to 1, *i.e.* $\alpha \triangleq \big(?E_\alpha; \alpha_1\big)^*$. The reasoning can be applied for arbitrary n and m.

We assume that we have a proof tree for the sequent $\mathcal{E}, A_\alpha \vdash [(?E_\alpha; \alpha_1)^*]G_\alpha$. The proof tree is of the form

$$\dfrac{\dfrac{\Pi_{Init_\alpha}}{\mathcal{E}, A_\alpha \vdash G_\alpha} \qquad \dfrac{\Pi_{Step_\alpha}}{\mathcal{E}, A_\alpha \vdash \forall^\alpha(G_\alpha \rightarrow [?E_\alpha; \alpha_1]G_\alpha)}}{\mathcal{E}, A_\alpha \vdash [(?E_\alpha; \alpha_1)^*]G_\alpha}\,[\text{Ind}]$$

We apply the induction rule and obtain two goals, the initial step $\mathcal{E}, A_\alpha \vdash G_\alpha$ and the induction step $\mathcal{E}, A_\alpha \vdash \forall^{\alpha\beta}(G_\alpha \to [?E_\alpha;\alpha_1]G_\alpha)$. We assume that they are closed by the branches Π_{Init_α} and Π_{Step_α}. We have a similar proof tree for the sequent $\mathcal{E}, A_\beta \vdash [(?E_\beta;\beta_1)^*]G_\beta$ with the branches Π_{Init_β} and Π_{Step_β}.

We want to derive a proof tree of the sequent $\mathcal{E}, A_\alpha, A_\beta \vdash [\big((?E_\alpha;\alpha_1) \cup (?E_\beta;\beta_1)\big)^*](E_\alpha \to G_\alpha \wedge E_\beta \to G_\beta)$. We use the notation $G_{\alpha\beta} \triangleq E_\alpha \to G_\alpha \wedge E_\beta \to G_\beta$ to shorten the representation of the guarantees. We first apply the induction rule.

$$\dfrac{\mathcal{E}, A_\alpha, A_\beta \vdash G_{\alpha\beta} \qquad \mathcal{E}, A_\alpha, A_\beta \vdash \forall^{\alpha\beta}\big(G_{\alpha\beta} \to [(?E_\alpha;\alpha_1) \cup (?E_\beta;\beta_1)]G_{\alpha\beta}\big)}{\mathcal{E}, A_\alpha, A_\beta \vdash [\big((?E_\alpha;\alpha_1) \cup (?E_\beta;\beta_1)\big)^*]G_{\alpha\beta}}\,[\text{Ind}]$$

We first consider the left premise which corresponds to the initial step. We unfold the notation $G_{\alpha\beta}$ and separate the conjunction to finally retrieve our assumptions Π_{Init_α} and Π_{Init_β} to close the proof tree.

$$\dfrac{\dfrac{\dfrac{\dfrac{\Pi_{Init_\alpha}}{\mathcal{E}, A_\alpha, A_\beta, E_\alpha \vdash G_\alpha}}{\mathcal{E}, A_\alpha, A_\beta \vdash E_\alpha \to G_\alpha}\to_r \qquad \dfrac{\dfrac{\Pi_{Init_\beta}}{\mathcal{E}, A_\alpha, A_\beta, E_\beta \vdash G_\beta}}{\mathcal{E}, A_\alpha, A_\beta \vdash E_\beta \to G_\beta}\to_r}{\mathcal{E}, A_\alpha, A_\beta \vdash E_\alpha \to G_\alpha \wedge E_\beta \to G_\beta}\wedge_r}{\mathcal{E}, A_\alpha, A_\beta \vdash G_{\alpha\beta}}\text{Unfold } G_{\alpha\beta}$$

We now consider the second premise which corresponds to the induction step. We want to derive a proof tree for the sequent $\mathcal{E}, A_\alpha, A_\beta \vdash \forall^{\alpha\beta}\big(G_{\alpha\beta} \to [(?E_\alpha;\alpha_1) \cup (?E_\beta;\beta_1)]G_{\alpha\beta}\big)$. We apply the skolemization rule $\forall_r$. As in the previous section, we denote with the superscript $^\alpha$ that a formula φ has seen all the occurrences of bound variables of α replaced by fresh variables. Thus $G_\alpha^{\alpha\beta}$ means that every occurrences of bound variables of both α and β in G_α have been replaced by fresh variables.

$$\dfrac{\dfrac{\mathcal{E}, A_\alpha, A_\beta \vdash (E_\alpha^{\alpha\beta} \to G_\alpha^{\alpha\beta} \wedge E_\beta^{\alpha\beta} \to G_\beta^{\alpha\beta}) \to [(?E_\alpha;\alpha_1)^{\alpha\beta} \cup (?E_\beta;\beta_1)^{\alpha\beta}](E_\alpha^{\alpha\beta} \to G_\alpha^{\alpha\beta} \wedge E_\beta^{\alpha\beta} \to G_\beta^{\alpha\beta})}{\mathcal{E}, A_\alpha, A_\beta \vdash \forall^{\alpha\beta}\big((E_\alpha \to G_\alpha \wedge E_\beta \to G_\beta) \to [(?E_\alpha;\alpha_1) \cup (?E_\beta;\beta_1)](E_\alpha \to G_\alpha \wedge E_\beta \to G_\beta)\big)}\forall_r}{\mathcal{E}, A_\alpha, A_\beta \vdash \forall^{\alpha\beta}\big(G_{\alpha\beta} \to [(?E_\alpha;\alpha_1) \cup (?E_\beta;\beta_1)]G_{\alpha\beta}\big)}\text{Unfold } G_{\alpha\beta}$$

We can apply the separation lemma (cf Lemma 1), thanks to the condition (b), to replace $G_\alpha^{\alpha\beta}$ by G_α^α. The variables of $G\alpha$ that may be captured by the quantification on bound variables of β are already captured by the quantification on bound variables of β. Idem for the replacement of $G_\beta^{\alpha\beta}$ by G_β^β. We apply the right implication rule $\to_r$ to the sequent $\mathcal{E}, A_\alpha, A_\beta \vdash (E_\alpha^{\alpha\beta} \to G_\alpha^\alpha \wedge E_\beta^{\alpha\beta} \to G_\beta^\beta) \to [(?E_\alpha;\alpha_1)^{\alpha\beta} \cup (?E_\beta;\beta_1)^{\alpha\beta}](E_\alpha^{\alpha\beta} \to G_\alpha^\alpha \wedge E_\beta^{\alpha\beta} \to G_\beta^\beta)$. We then split the program $(?E_\alpha;\alpha_1)^{\alpha\beta} \cup (?E_\beta;\beta_1)^{\alpha\beta}$ in two with the non-deterministic choice rule $[\cup]$.

$$\underbrace{\mathcal{E}, A_\alpha, A_\beta, E_\alpha^{\alpha\beta} \to G_\alpha^\alpha \wedge E_\beta^{\alpha\beta} \to G_\beta^\beta \vdash [(?E_\beta;\beta_1)^{\alpha\beta}](E_\alpha^{\alpha\beta} \to G_\alpha^\alpha \wedge E_\beta^{\alpha\beta} \to G_\beta^\beta)}$$

$$\underbrace{\mathcal{E}, A_\alpha, A_\beta, E_\alpha^{\alpha\beta} \to G_\alpha^\alpha \wedge E_\beta^{\alpha\beta} \to G_\beta^\beta \to G_\beta^{\alpha\beta} \vdash [(?E_\alpha;\alpha_1)^{\alpha\beta}](E_\alpha^{\alpha\beta} \to G_\alpha^\alpha \wedge E_\beta^{\alpha\beta} \to G_\beta^\beta)}$$

$$\dfrac{\dfrac{}{\mathcal{E}, A_\alpha, A_\beta, E_\alpha^{\alpha\beta} \to G_\alpha^\alpha \wedge E_\beta^{\alpha\beta} \to G_\beta^\beta \vdash [(?E_\alpha;\alpha_1)^{\alpha\beta} \cup (?E_\beta;\beta_1)^{\alpha\beta}](E_\alpha^{\alpha\beta} \to G_\alpha^\alpha \wedge E_\beta^{\alpha\beta} \to G_\beta^\beta)}[\cup]}{\mathcal{E}, A_\alpha, A_\beta \vdash (E_\alpha^{\alpha\beta} \to G_\alpha^\alpha \wedge E_\beta^{\alpha\beta} \to G_\beta^\beta) \to [(?E_\alpha;\alpha_1)^{\alpha\beta} \cup (?E_\beta;\beta_1)^{\alpha\beta}](E_\alpha^{\alpha\beta} \to G_\alpha^\alpha \wedge E_\beta^{\alpha\beta} \to G_\beta^\beta)} \to_r$$

We have two cases to consider. We only consider the first one, the sequent $\mathcal{E}, A_\alpha, A_\beta, E_\alpha^{\alpha\beta} \to G_\alpha^\alpha \wedge E_\beta^{\alpha\beta} \to G_\beta^\beta \vdash [(?E_\alpha;\alpha_1)^{\alpha\beta}](E_\alpha^{\alpha\beta} \to G_\alpha^\alpha \wedge E_\beta^{\alpha\beta} \to G_\beta^\beta)$; the second case is similar.

We split the invariant $(E_\alpha^{\alpha\beta} \to G_\alpha^\alpha \wedge E_\beta^{\alpha\beta} \to G_\beta^\beta)$ with the rule [BoxAnd] and consider separately each premise.

$$\underbrace{\mathcal{E}, A_\alpha, A_\beta, E_\alpha^{\alpha\beta} \to G_\alpha^\alpha, E_\beta^{\alpha\beta} \to G_\beta^\beta \vdash [(?E_\alpha;\alpha_1)^{\alpha\beta}](E_\beta^{\alpha\beta} \to G_\beta^\beta)}$$

$$\dfrac{\mathcal{E}, A_\alpha, A_\beta, E_\alpha^{\alpha\beta} \to G_\alpha^\alpha, E_\beta^{\alpha\beta} \to G_\beta^\beta \vdash [(?E_\alpha;\alpha_1)^{\alpha\beta}](E_\alpha^{\alpha\beta} \to G_\alpha^\alpha)}{\mathcal{E}, A_\alpha, A_\beta, E_\alpha^{\alpha\beta} \to G_\alpha^\alpha, E_\beta^{\alpha\beta} \to G_\beta^\beta \vdash [(?E_\alpha;\alpha_1)^{\alpha\beta}](E_\alpha^{\alpha\beta} \to G_\alpha^\alpha \wedge E_\beta^{\alpha\beta} \to G_\beta^\beta)} [\text{BoxAnd}]$$

We close the first premise $\mathcal{E}, A_\alpha, A_\beta, E_\alpha^{\alpha\beta} \to G_\alpha^\alpha, E_\beta^{\alpha\beta} \to G_\beta^\beta \vdash [(?E_\alpha;\alpha_1)^{\alpha\beta}](E_\alpha^{\alpha\beta} \to G_\alpha^\alpha)$ with the branch Π_{Step_α}. For the second premise, we make use of the assumption that the events E_α and E_β are exclusive.

First, notice that $(?E_\alpha;\alpha_1)^{\alpha\beta}$ is equivalent to $?E_\alpha^{\alpha\beta};\alpha_1^{\alpha\beta}$. We apply the sequential composition rule [;] followed by the text rule [?]. We apply then the rule $\to_r$ to pass the left-hand side of the implication in the hypothesis.

$$\dfrac{\dfrac{\dfrac{\mathcal{E}, A_\alpha, A_\beta, E_\alpha^{\alpha\beta} \to G_\alpha^\alpha, E_\beta^{\alpha\beta} \to G_\beta^\beta, E_\alpha^{\alpha\beta} \vdash [\alpha_1^{\alpha\beta}](E_\beta^{\alpha\beta} \to G_\beta^\beta)}{\mathcal{E}, A_\alpha, A_\beta, E_\alpha^{\alpha\beta} \to G_\alpha^\alpha, E_\beta^{\alpha\beta} \to G_\beta^\beta \vdash E_\alpha^{\alpha\beta} \to [\alpha_1^{\alpha\beta}](E_\beta^{\alpha\beta} \to G_\beta^\beta)} \to_r}{\mathcal{E}, A_\alpha, A_\beta, E_\alpha^{\alpha\beta} \to G_\alpha^\alpha, E_\beta^{\alpha\beta} \to G_\beta^\beta \vdash [?E_\alpha^{\alpha\beta}][\alpha_1^{\alpha\beta}](E_\beta^{\alpha\beta} \to G_\beta^\beta)} [?]}{\mathcal{E}, A_\alpha, A_\beta, E_\alpha^{\alpha\beta} \to G_\alpha^\alpha, E_\beta^{\alpha\beta} \to G_\beta^\beta \vdash [?E_\alpha^{\alpha\beta};\alpha_1^{\alpha\beta}](E_\beta^{\alpha\beta} \to G_\beta^\beta)} [;]$$

We symbolically execute $\alpha_1^{\alpha\beta}$. Since E_β does not refer to bound variables of α, the symbolic execution of $\alpha_1^{\alpha\beta}$ does not modify $E_\beta^{\alpha\beta}$. We have to derive a proof tree for the sequent $\mathcal{E}, A_\alpha, A_\beta, E_\alpha^{\alpha\beta} \to G_\alpha^\alpha, E_\beta^{\alpha\beta} \to G_\beta^\beta, E_\alpha^{\alpha\beta} \vdash E_\beta^{\alpha\beta} \to (G_\beta^\beta)^{\alpha_1^{\alpha\beta}}$.

We pass the event $E_\beta^{\alpha\beta}$ in the left-hand side of the sequent with the rule $\to_r$ and conclude by *ex falso quod libet* (rule $\perp$). Indeed, we have that E_α and E_β are exclusive, thus $E_\alpha^{\alpha\beta}$ and $E_\beta^{\alpha\beta}$ also.

$$\cfrac{\overline{\mathcal{E}, A_\alpha, A_\beta, E_\alpha^{\alpha\beta} \to G_\alpha^\alpha, E_\beta^{\alpha\beta} \to G_\beta^\beta, E_\alpha^{\alpha\beta}, E_\beta^{\alpha\beta} \vdash (G_\beta^\beta)^{\alpha_1^{\alpha\beta}}}\ \bot}{\mathcal{E}, A_\alpha, A_\beta, E_\alpha^{\alpha\beta} \to G_\alpha^\alpha, E_\beta^{\alpha\beta} \to G_\beta^\beta, E_\alpha^{\alpha\beta} \vdash E_\beta^{\alpha\beta} \to (G_\beta^\beta)^{\alpha_1^{\alpha\beta}}}\ \to_r$$

□

Example 57 (Contract for the error mode). *The contract* (A_{err}, G_{err}) *for the* `Error` *component is:*

$$\begin{cases} A_{err}: & \\ G_{err}: & fin = 0 \\ & fout = 0 \end{cases}$$

It is simple to verify that `Error` *satisfies its contract. As we consider the two components without a context, we can drop the assumption on the outputs of the water-level, we restrict the contract of* `Controller` *as following:*

$$\begin{cases} A_{ctrl}: & \\ G_{ctrl}: & wlm \leqslant 3.5 \to fin = 1 \\ & 6.5 \leqslant wlm_1 \to fin = 0 \\ & (3.5 \geqslant wlm_1 \geqslant 6.5) \to (fin = 0 \vee fin = 1) \end{cases}$$

Example 58 (Composition of `Error` and `Controller` components). *The event* $error = 0$ *and* $error = 1$ *are exclusive.* G_{err} *does not refer to outputs of* `Controller` *other that* fin *which is the common output. Also,* G_{ctrl} *does not refer to outputs of* `Error` *other that* fin*. The condition (c) is also verified trivially.*

Thus we have that the component `ControllerError` *resulting from the composition of* `Controller` *and* `Error` *satisfies the contract* $\big(A_{err} \wedge A_{ctrl}, (error = 0 \to A_{ctrl}) \wedge (error = 1 \to A_{err})\big)$*.*

Conclusion

Compared to the previous theorem on parallel composition, we drop the assumptions that the outputs of the two components must be different, and still retain that the compound component satisfies the conjunction of respective guarantees.

We have presented how to handle two modes, but we think it is easy to generalize. To retain the associativity of the parallel composition operator $\circ$, we must be careful that each added event is mutually exclusive with all the other existing events. We think that a similar mechanism can be extended to components of the general form with the event being incorporated in the evolution of the continuous evolution. Yet, a mode is typically two possible functioning flow of a system and is something decided by a human.

When we reason on modes, we do not want to loose the reasoning on temporal constraints as presented in Section 4.2. We think it is possible to integrate both aspects, but it requires extra carefulness to mix theses methods. For example, the notification variable t_{ctrl} must be shared with the component `Error`.

Fault-tolerance may also be handled using our mechanism of modes. The appearance of a fault is modeled as one mode of functioning and the composition is carried accordingly.

4.4 A Causal Composition operator

When building a system from components, we may want to enforce a special ordering between two components because they have a causal relation, for example the composition of a sensor and the associated monitor. The parallel composition operator is not fitted for that. In this section, we present how we adapt it by adding ordering constraints to obtain a so-called a causal composition operator.

We work in the framework of timed parallel composition defined in Section 4.2. We make advantage of the notion of notification already introduced. Hence, one is able to freely combine our causal composition operator with the timed parallel composition operator. We exemplify the causal composition by breaking the water-level controller into three components, a sensor, a monitor and an actuator. We want the sensor to run before the monitor since the monitor needs value from the sensor to decide what to do. The actuator have to run after the monitor since it needs the decision of the monitor to actuate on the system. Such decomposition has the advantage that each component performs exactly one task, but our previous parallel composition operator was not fitted for that.

4.4.1 Modular Modelling

We define the causal composition operator for discrete components. To our knowledge, it is not clear if an extension to continuous systems makes sense. Indeed, a design justification for the ordering of two continuous evolution is not obvious. But it may be interesting for the structuring of the proof for a continuous evolution with different stages, *e.g.* the growth of a bacterial population is first a lag phase, then exponential, followed by a stationary phase and then a death phase as the resources rarefy.

Recall that the behavior α of a timed discrete component `A` and execution period Δ_α is of the form $\big((?t \leqslant t_{\alpha_1} + \Delta_\alpha; \alpha_1; t_{\alpha_1} := t) \cup \ldots \cup (?t \leqslant t_{\alpha_n} + \Delta_\alpha; \alpha_n; t_{\alpha_n} := t)\big)^*$. α_i are functional behaviors of sub-components of `A`. The test $?t \leqslant t_{\alpha_i} + \Delta_\alpha$ requires that at most Δ_α units of time have passed since the last execution of α. The assignment $t_{\alpha_i} := t$ remembers the instant of the last execution of α.

Example 59 (Behavior of the water-level sensor)**.** *The functional behavior of the water-level sensor component, denoted* `Sensor`*, is* $wlm := wl$*. Its execution period is* 0.01 *seconds and the notification is the variable* t_{sens}*. Thus, its behavior is:*

$$(?t \leqslant t_{sens} + 0.01; wlm := wl; t_{sens} := t)^*$$

Example 60 (Behavior of the water-level monitor)**.** *The functional behavior of the water-level monitor component, denoted* `Monitor` *is the functional behavior of the controller component as in Subsection 3.5.3, but without the sensing part. It is* $(?wlm_1 \geqslant 6.5; fin := 0) \cup (?wlm_1 \leqslant 3.5; fin := 1)$ *where the inlet valve is open or closed according to the value send by the sensor. Its execution period is* 0.02 *seconds. Thus, the behavior of the water-level monitor is:*

$$\big(?t \leqslant t_{ctrl} + 0.02; (?wlm \geqslant 6.5; fin := 0) \cup (?wlm \leqslant 3.5; fin := 1); t_{ctrl} := t\big)^*$$

The execution period of a compound component obtained by the causal composition of two components `A` and `B` is the sum of respective execution periods.

Definition 49. *The component resulting from the causal composition of two discrete components* `A` *and* `B` *is:*

```
Compound Component: AB
Period:
 Δ_α + Δ_β
Inputs:
 A.inputs ∪ B.inputs
Assumptions:
 A_α ∧ A_β
Outputs:
 A.outputs ∪ B.outputs
Guarantees:
 G_α ∧ G_β
Functional behavior:
 α ⊙ β
```

We first present the definition for the case of two atomic discrete components to clearly identify the additions enforcing the ordering. We generalize it in the Definition 51. We proceed as for the timed parallel composition with the update of the execution period, but we also add ordering constraints in the test in front of the functional behavior.

Definition 50 (Causal composition of discrete atomic component)**.** *Let α and β be two functional behaviors of discrete atomic components* `A` *and* `B` *with respective periods Δ_α and Δ_β. Their causal composition $\alpha \odot \beta$ is given by:*

$$\begin{pmatrix} (?t_\beta < t \leqslant t_\alpha + \Delta_\alpha + \Delta_\beta; \alpha; t_\alpha := t) \\ \cup\ (?t \leqslant t_\beta + \Delta_\alpha + \Delta_\beta \wedge t_\beta \leqslant t_\alpha; \beta; t_\beta := t) \end{pmatrix}^*$$

As for the timed parallel composition, we replace the occurrences of Δ_α and Δ_β by the sum $\Delta_\alpha + \Delta_\beta$. We add two supplementary formulas: $t_\beta < t$ in the guard of α and $t_\beta \leqslant t_\alpha$ in the guard of β. The second formula constraint β to wait that α has executed, *i.e.* that the value of t_α is greater meaning that it has executed more recently. It rules out traces where β executes before α.

The formula $t_\beta < t$ requires that a non-zero amount of time has passed since the last execution of β before executing again α. We have to remember that we are in a context where we model systems that take real time. It is thus not too restrictive to add the assumption that a small lapse of time have passed since the last execution.

Example 61 (Sensor-Monitor)**.** *We causally compose the* `Sensor` *component with the* `Monitor` *component. The resulting component* `SensorMonitor` *has the following behavior. The resulting execution period is* $0.01 + 0.02 = 0.03$.

$$\begin{pmatrix} (?t \leqslant t_{sens} + 0.03; wlm := wl; t_{sens} := t) \\ \cup\ (?t \leqslant t_{ctrl} + 0.03; (?wlm \geqslant 6.5; fin := 0) \cup (?wlm \leqslant 3.5; fin := 1); t_{ctrl} := t) \end{pmatrix}^*$$

We generalize it for discrete general components. Remember that the canonical form of a discrete behavior with execution period Δ_α is $\alpha \triangleq \big(\bigcup_{1 \leqslant i \leqslant n} (?t \leqslant t_{\alpha_i} + \Delta_\alpha; \alpha_i; t_{\alpha_i} := t)\big)^*$. The α_i are functional behaviors of discrete atomic sub-components.

But the component may result from the causal composition of several of its sub-components. We thus have to add the formula $Lapse_{\alpha_i}$ and Ord_{α_i} accounting respectively for the forbidding

of a instant execution of the first component after the execution of the second component and for the possible already existing ordering constraints. They may be equal to $\top$. In our definition with discrete atomic components, we have $Lapse_\alpha \triangleq t_\beta < t$, $Lapse_\beta \triangleq \top$, $Ord_\alpha \triangleq \top$ and $Ord_\beta \triangleq t_\beta \leqslant t_\alpha$.

Definition 51. *(Discrete causal composition). Let α and β be two discrete behaviors of components* `A` *and* `B` *with execution periods Δ_α and Δ_β. They respect the canonical form $\big(\bigcup_{1\leqslant i\leqslant n}(?Lapse_{\alpha_i} \wedge t \leqslant t_{\alpha_i} + \Delta_\alpha \wedge Ord_{\alpha_i}; \alpha_i; t_{\alpha_i} := t)\big)^*$ for the behavior α and we have $\big(\bigcup_{1\leqslant j\leqslant m}(?Lapse_{\beta_j} \wedge t \leqslant t_{\beta_j} + \Delta_\beta \wedge Ord_{\beta_j}; \beta_j; t_{\beta_j} := t)\big)^*$ for the behavior β.*

The causal composition $\alpha \odot \beta$ is defined by:

$$\begin{pmatrix} \bigcup_{1\leqslant i\leqslant n}\big(?(\bigwedge_{1\leqslant j\leqslant m} t_{\beta_j} < t) \wedge Lapse_{\alpha_i} \wedge t \leqslant t_{\alpha_i} + \Delta_\alpha + \Delta_\beta \wedge Ord_{\alpha_i}; \alpha_i; t_{\alpha_i} := t\big) \\ \cup \bigcup_{1\leqslant j\leqslant m}\big(?Lapse_{\beta_j} \wedge t \leqslant t_{\beta_j} + \Delta_\alpha + \Delta_\beta \wedge Ord_{\beta_j} \wedge (\bigwedge_{1\leqslant i\leqslant n} t_{\alpha_i} \geqslant t_{\beta_j}); \beta_j; t_{\beta_j} := t\big) \end{pmatrix}^*$$

We add the constraint $(\bigwedge_{1\leqslant j\leqslant m} t_{\beta_j} < t)$ in front of each sub-behaviors α_i of α. Since α may result from previous parallel composition, and thus be made of several elements that are independent, we have to forbid each of them to execute instantaneously after the execution of β. We also add the ordering constraint $(\bigwedge_{1\leqslant i\leqslant n} t_{\alpha_i} \geqslant t_{\beta_j})$ in front of each sub-behaviors β_j of β. Indeed, we have to constraint each of its sub-components to start before the end of execution of α.

4.4.2 Algebraic properties

We show in this subsection that we retain the associativity property, essential to achieve a truly modular component-based approach. We lose the property of commutativity, but it is expected since we want to ensure a particular order between two components.

Proposition 8 (Non-commutativity and associativity)**.** *Let α, β and γ be behaviors of discrete components* `A`*,* `B` *and* `C`*, of respective execution periods Δ_α, Δ_β and Δ_γ.*

$$\begin{array}{lcll} \alpha \odot \beta & \neq & \beta \odot \alpha & \textit{(Non-commutativity)} \\ (\alpha \odot \beta) \odot \gamma & = & \alpha \odot (\beta \odot \gamma) & \textit{(Associativity)} \end{array}$$

The *associativity* property ensures that the modularity is preserved with the addition of the causal composition operator.

Proof. Recall that the behavior α respects the following form

$$\Big(\bigcup_{1\leqslant i\leqslant n}\big(?Lapse_{\alpha_i\beta} \wedge (t \leqslant t_{\alpha_i} + \Delta_\alpha)_{\Delta_\alpha}^{\Delta_\alpha+\Delta_\beta} \wedge Ord_{\alpha_i}; \alpha_i; t_{\alpha_i} := t\big)\Big)^*$$

The behaviors β and γ respect a similar form.

For the proof of non-commutativity, it is a negative result and we just have to exhibit a counter-example to associativity. For that, we chose to reason with two discrete atomic components to be in the case of Definition 50, simpler to handle. It means that $n = m = 1$ and thus α is of the form $(?t \leqslant t_{\alpha_1} + \Delta_\alpha; \alpha_1; t_{\alpha_1} := t)^*$ and β is $(?t \leqslant t_{\beta_1} + \Delta_\beta; \beta_1; t_{\beta_1} := t)^*$.

We show that there exists reachable states of $\alpha \odot \beta$ that are not reachable by $\beta \odot \alpha$. We have:

$$\begin{array}{rcl}\alpha \odot \beta & \triangleq & \big((?t_{\beta_1} < t \leqslant t_{\alpha_1} + \Delta_\alpha + \Delta_\beta; \alpha_1; t_{\alpha_1} := t) \\ & & \cup (?t \leqslant t_{\beta_1} + \Delta_\alpha + \Delta_\beta \wedge t_{\beta_1} \leqslant t_{\alpha_1}; \beta_1; t_{\beta_1} := t)\big)^*\end{array}$$

and on the other side:

$$\begin{array}{rcl}\beta \odot \alpha & \triangleq & \big((?t_{\alpha_1} < t \leqslant t_{\beta_1} + \Delta_\beta + \Delta_\alpha; \beta_1; t_{\beta_1} := t) \\ & & \cup (?t \leqslant t_{\alpha_1} + \Delta_\beta + \Delta_\alpha \wedge t_{\alpha_1} \leqslant t_{\beta_1}; \alpha_1; t_{\alpha_1} := t)\big)^*\end{array}$$

In the first case, a sole execution of α is possible, thus $\rho(?t_\beta < t \leqslant t_\alpha + \Delta_\alpha + \Delta_\beta; \alpha; t_\alpha := t) \subseteq \rho(\alpha \odot \beta)$. But in the second case, a sole execution of α is not possible. We must have β executed before executing α, thus $\rho(?t_\beta < t \leqslant t_\alpha + \Delta_\alpha + \Delta_\beta; \alpha; t_\alpha := t) \nsubseteq \rho(\beta \odot \alpha)$.

For the associativity property, we unfold the definition of each successive causal composition. We denote the formula $(\bigwedge_{1 \leqslant j \leqslant m} t_{\beta_j} < t) \wedge Lapse_{\alpha_i}$ by $Lapse_{\alpha_i\beta}$. In the similar manner, we denote the formula $Ord_{\beta_j} \wedge (\bigwedge_{1 \leqslant i \leqslant n} t_{\alpha_i} \geqslant t_{\beta_j})$ by $Ord_{\alpha\beta_j}$.

We unfold twice the definition of $\odot$ for $(\alpha \odot \beta) \odot \gamma$.

$$\begin{array}{rl}
& (\alpha \odot \beta) \odot \gamma \\
= & \Big(\Big(\bigcup_{1 \leqslant i \leqslant n} \big(?Lapse_{\alpha_i\beta} \wedge t \leqslant t_{\alpha_i} + \Delta_\alpha + \Delta_\beta \wedge Ord_{\alpha_i}; \alpha_i; t_{\alpha_i} := t\big) \\
& \cup \bigcup_{1 \leqslant j \leqslant m} \big(?Lapse_{\beta_j} \wedge t \leqslant t_{\beta_j} + \Delta_\alpha + \Delta_\beta \wedge Ord_{\alpha\beta_j}; \beta_j; t_{\beta_j} := t\big)\Big)^*\Big) \odot \gamma \\
= & \Big(\bigcup_{1 \leqslant i \leqslant n} \big(?Lapse_{\alpha_i\beta} \wedge Lapse_{\alpha_i\gamma} \wedge t \leqslant t_{\alpha_i} + (\Delta_\alpha + \Delta_\beta) + \Delta_\gamma \wedge Ord_{\alpha_i}; \alpha_i; t_{\alpha_i} := t\big) \\
& \cup \bigcup_{1 \leqslant j \leqslant m} \big(?Lapse_{\beta_j\gamma} \wedge t \leqslant t_{\beta_j} + (\Delta_\alpha + \Delta_\beta) + \Delta_\gamma \wedge Ord_{\alpha\beta_j}; \beta_j; t_{\beta_j} := t\big) \\
& \cup \bigcup_{1 \leqslant k \leqslant p} \big(?Lapse_{\gamma_k} \wedge t \leqslant t_{\gamma_k} + (\Delta_\alpha + \Delta_\beta) + \Delta_\gamma \wedge Ord_{\alpha\gamma_k} \wedge Ord_{\beta\gamma_k}; \gamma_k; t_{\gamma_k} := t\big)\Big)^*
\end{array}$$

We expand the definition for $\alpha \odot (\beta \odot \gamma)$

$$\begin{array}{rl}
& \alpha \odot (\beta \odot \gamma) \\
= & \alpha \odot \Big(\Big(\bigcup_{1 \leqslant j \leqslant m} \big(?Lapse_{\beta_j\gamma} \wedge t \leqslant t_{\beta_j} + \Delta_\beta + \Delta_\gamma \wedge Ord_{\beta_j}; \beta_j; t_{\beta_j} := t\big) \\
& \quad \cup \bigcup_{1 \leqslant k \leqslant p} \big(?Lapse_{\gamma_k} \wedge t \leqslant t_{\gamma_k} + \Delta_\beta + \Delta_\gamma \wedge Ord_{\beta\gamma_k}; \gamma_k; t_{\gamma_k} := t\big)\Big)^*\Big) \\
= & \Big(\bigcup_{1 \leqslant i \leqslant n} \big(?Lapse_{\alpha_i\beta} \wedge Lapse_{\alpha_i\beta} \wedge t \leqslant t_{\alpha_i} + \Delta_\alpha + (\Delta_\beta + \Delta_\gamma) \wedge Ord_{\alpha_i}; \alpha_i; t_{\alpha_i} := t\big) \\
& \cup \bigcup_{1 \leqslant j \leqslant m} \big(?Lapse_{\beta_j\gamma} \wedge t \leqslant t_{\beta_j} + \Delta_\alpha + (\Delta_\beta + \Delta_\gamma) \wedge Ord_{\alpha\beta_j}; \beta_j; t_{\beta_j} := t\big) \\
& \cup \bigcup_{1 \leqslant k \leqslant p} \big(?Lapse_{\gamma_k} \wedge t \leqslant t_{\gamma_k} + \Delta_\alpha + (\Delta_\beta + \Delta_\gamma) \wedge Ord_{\alpha\gamma_k} \wedge Ord_{\beta\gamma_k}; \gamma_k; t_{\gamma_k} := t\big)\Big)^*
\end{array}$$

Since $(\Delta_\alpha + \Delta_\beta) + \Delta_\gamma = \Delta_\alpha + (\Delta_\beta + \Delta_\gamma)$, we conclude that $(\alpha \odot \beta) \odot \gamma = \alpha \odot (\beta \odot \gamma)$ because the unfolded definition are equivalent. □

4.4.3 Modular proof

We present a version of the Theorem 4 where we relax the condition that the guarantee of `B` must not refer to the outputs of `A`. The idea is to temporally characterize the execution of the component `B` (or of one of its sub-components). At this instant, we know that `A` has not executed, and we thus refers to outputs of `A`.

Recall that the behavior β of a discrete component `B` respects the following form $\big(\bigcup_{1\leqslant j\leqslant m}(?Lapse_{\beta_j} \wedge t \leqslant t_{\beta_j} + \Delta_\beta \wedge Ord_{\beta_j}; \beta_j; t_{\beta_j} := t)\big)^*$. It is characterized by the formula $\bigvee_{1\leqslant j\leqslant m} t_{\beta_j} = t$, hence the following theorem.

Theorem 12. *Let α and β be two behaviors of discrete components* `A` *and* `B` *with respective contracts (A_α, G_α) and (A_β, G_β) and respective execution periods Δ_α and Δ_β. Assume that we have a proof of $\mathcal{E}, A_\alpha \vdash [\alpha]G_\alpha$ and $\mathcal{E}, A_\beta \vdash [\beta]G_\beta$. Then, under the conditions:*

(a) $BV(\alpha) \cap BV(\beta) = \varnothing$,

(b) $BV(\beta) \cap FV(G_\alpha) = \varnothing$,

(c_1) $A_\alpha \vdash \forall^\beta(G_\beta \to A_\alpha)$

(c_2) $A_\beta \vdash \forall^\alpha(G_\alpha \to A_\beta)$

We have a proof of:

$$\mathcal{E}, A_\alpha, A_\beta \vdash [\alpha \odot \beta]\big(G_\alpha \wedge (\bigvee_{1\leqslant j\leqslant m} t_{\beta_j} = t) \to G_\beta)\big)$$

Proof. For simplicity, we assume that the behaviors α and β are atomic. It is straightforward to generalize. We recall that the behaviors α and β respect the following canonical form $(?t \leqslant t_{\alpha_1} + \Delta_\alpha + \Delta_\beta; \alpha_1; t_{\alpha_1} := t)^*$ and $(?t \leqslant t_{\beta_1} + \Delta_\alpha + \Delta_\beta; \beta_1; t_{\beta_1} := t)^*$. G_β is of the form $t_{\beta_1} = t \to \varphi_{\beta_1}$.

By assumption, we have a proof tree of the sequents $\mathcal{E}, A_\alpha \vdash [(?t \leqslant t_{\alpha_1} + \Delta_\alpha + \Delta_\beta; \alpha_1; t_{\alpha_1} := t)^*]G_\alpha$ and $\mathcal{E}, A_\beta \vdash [(?t \leqslant t_{\beta_1} + \Delta_\alpha + \Delta_\beta; \beta_1; t_{\beta_1} := t)^*](t_{\beta_1} = t \to G_\beta)$.

The proof tree of $\mathcal{E}, A_\alpha \vdash [(?t \leqslant t_{\alpha_1} + \Delta_\alpha + \Delta_\beta; \alpha_1; t_{\alpha_1} := t)^*]G_\alpha$ is of the following form:

$$\dfrac{\dfrac{\Pi_{Init_\alpha}}{\mathcal{E}, A_\alpha \vdash G_\alpha} \qquad \dfrac{\Pi_{Step_\alpha}}{\mathcal{E}, A_\alpha \vdash \forall^\alpha(G_\alpha \to [?t \leqslant t_{\alpha_1} + \Delta_\alpha; \beta_1; t_{\alpha_1} := t]G_\alpha)}}{\mathcal{E}, A_\alpha \vdash [(?t \leqslant t_{\alpha_1} + \Delta_\alpha; \alpha_1; t_{\alpha_1} := t)^*]G_\alpha}\,[\text{Ind}]$$

We have a similar proof tree for the sequent $\mathcal{E}, A_\beta \vdash [(?t \leqslant t_{\beta_1} + \Delta_\alpha + \Delta_\beta; \beta_1; t_{\beta_1} := t)^*](t_{\beta_1} = t \to G_\beta)$. Using the branches Π_{Init_α}, Π_{Init_β}, Π_{Step_α} and Π_{Step_β}, we derive a proof tree for the sequent $\mathcal{E}, A_\alpha, A_\beta \vdash [\alpha_1 \odot \beta_1](G_\alpha \wedge t_{\beta_1} = t \to G_\beta)$.

We unfold the definition of $\alpha \odot \beta$ before applying the induction rule [Ind]. We adopt the notation $body_\alpha \triangleq (?t_{\beta_1} < t \leqslant t_{\alpha_1} + \Delta_\alpha + \Delta_\beta; \alpha_1; t_{\alpha_1} := t)$ and $body_\beta \triangleq (?t \leqslant t_{\beta_1} + \Delta_\alpha + \Delta_\beta \wedge t_{\beta_1} \leqslant t_{\alpha_1}; \beta_1; t_{\beta_1} := t)$.

$$\dfrac{\dfrac{\mathcal{E}, A_\alpha, A_\beta \vdash G_\alpha \wedge t_{\beta_1} = t \to G_\beta \qquad \mathcal{E}, A_\alpha, A_\beta \vdash \forall^{\alpha\beta}\big((G_\alpha \wedge t_{\beta_1} = t \to G_\beta) \to [body_\alpha \cup body_\beta](G_\alpha \wedge t_{\beta_1} = t \to G_\beta)\big)}{\mathcal{E}, A_\alpha, A_\beta \vdash [\big(body_\alpha \cup body_\beta\big)^*](G_\alpha \wedge t_{\beta_1} = t \to G_\beta)}\,[\text{Ind}]}{\mathcal{E}, A_\alpha, A_\beta \vdash [\alpha \odot \beta](G_\alpha \wedge t_{\beta_1} = t \to G_\beta)}\,\text{Unfolding}$$

The two premises correspond to the initial step and the induction step. We first consider the case of the initial step. We split the conjunction using the rule $\wedge_r$. We close each premise with the branches Π_{Init_α} and Π_{Init_β}.

$$\dfrac{\dfrac{\Pi_{Init_\alpha}}{\mathcal{E}, A_\alpha, A_\beta \vdash G_\alpha} \quad \dfrac{\Pi_{Init_\beta}}{\mathcal{E}, A_\alpha, A_\beta \vdash t_{\beta_1} = t \to G_\beta}}{\mathcal{E}, A_\alpha, A_\beta \vdash G_\alpha \wedge t_{\beta_1} = t \to G_\beta} \wedge_r$$

It remains the induction step to consider. We first apply the rule $\forall_r$ to replace occurrences of bound variables of α and β by fresh variables. We separate then the program by applying the non-deterministic choice rule $[\cup]$.

$$\underbrace{\mathcal{E}, A_\alpha, A_\beta, G_\alpha^{\alpha\beta} \wedge (t_{\beta_1} = t \to G_\beta)^{\alpha\beta} \vdash [body_\beta^{\alpha\beta}](G_\alpha^{\alpha\beta} \wedge (t_{\beta_1} = t \to G_\beta)^{\alpha\beta})}$$

$$\underbrace{\mathcal{E}, A_\alpha, A_\beta, G_\alpha^{\alpha\beta} \wedge (t_{\beta_1} = t \to G_\beta)^{\alpha\beta} \vdash [body_\alpha^{\alpha\beta}](G_\alpha^{\alpha\beta} \wedge (t_{\beta_1} = t \to G_\beta)^{\alpha\beta})}$$

$$\dfrac{\dfrac{\dfrac{}{\mathcal{E}, A_\alpha, A_\beta, G_\alpha^{\alpha\beta} \wedge (t_{\beta_1} = t \to G_\beta)^{\alpha\beta} \vdash [body_\alpha^{\alpha\beta} \cup body_\beta^{\alpha\beta}](G_\alpha^{\alpha\beta} \wedge (t_{\beta_1} = t \to G_\beta)^{\alpha\beta})} [\cup]}{\mathcal{E}, A_\alpha, A_\beta \vdash (G_\alpha^{\alpha\beta} \wedge (t_{\beta_1} = t \to G_\beta)^{\alpha\beta}) \to [body_\alpha^{\alpha\beta} \cup body_\beta^{\alpha\beta}](G_\alpha^{\alpha\beta} \wedge (t_{\beta_1} = t \to G_\beta)^{\alpha\beta})} \to_r}{\mathcal{E}, A_\alpha, A_\beta \vdash \forall^{\alpha\beta}\big((G_\alpha \wedge (t_{\beta_1} = t \to G_\beta)) \to [body_\alpha \cup body_\beta](G_\alpha \wedge (t_{\beta_1} = t \to G_\beta))\big)} \forall_r$$

We have two cases to consider, the case where there is one execution of α and the case where there is one execution of β. The second case is similar to the reasoning in Theorem 4. This reasoning can not be applied to the first case since G_β may refer to bound variables of α and thus be modified.

We first unfold the notation $body_\alpha$. We split the guarantee in the sequent $\mathcal{E}, A_\alpha, A_\beta, G_\alpha^{\alpha\beta} \wedge (t_{\beta_1} = t \to G_\beta)^{\alpha\beta} \vdash [(?t_{\beta_1} < t \leqslant t_{\alpha_1} + \Delta_\alpha + \Delta_\beta; \alpha_1; t_{\alpha_1} := t)^{\alpha\beta}](G_\alpha^{\alpha\beta} \wedge (t_{\beta_1} = t \to G_\beta)^{\alpha\beta})$ with the rule [BoxAnd]. It yields two goals. Using the condition (b), we have that $G_\alpha^{\alpha\beta}$ is equivalent to G_α^{α}. The goal $\mathcal{E}, A_\alpha, A_\beta, G_\alpha^{\alpha} \wedge (t_{\beta_1} = t \to G_\beta)^{\alpha\beta} \vdash [(?t_{\beta_1} < t \leqslant t_{\alpha_1} + \Delta_\alpha + \Delta_\beta; \alpha_1; t_{\alpha_1} :=$

$t)^{\alpha\beta}]G_\alpha^\alpha$ is closed by the branch Π_{Step_α}.

$$\underbrace{\mathcal{E}, A_\alpha, A_\beta, G_\alpha^{\alpha\beta} \wedge G_\beta^{\alpha\beta} \vdash [(?t_{\beta_1} < t \leqslant t_{\alpha_1} + \Delta_\alpha + \Delta_\beta; \alpha_1; t_{\alpha_1} := t)^{\alpha\beta}]G_\beta^{\alpha\beta}}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\Pi_{Step_\alpha}}{\mathcal{E}, A_\alpha, A_\beta, G_\alpha^\alpha \wedge (t_{\beta_1} = t \to G_\beta)^{\alpha\beta} \vdash [(?t_{\beta_1} < t \leqslant t_{\alpha_1} + \Delta_\alpha + \Delta_\beta; \alpha_1; t_{\alpha_1} := t)^{\alpha\beta}]G_\alpha^\alpha}}{\mathcal{E}, A_\alpha, A_\beta, G_\alpha^{\alpha\beta} \wedge (t_{\beta_1} = t \to G_\beta)^{\alpha\beta} \vdash [(?t_{\beta_1} < t \leqslant t_{\alpha_1} + \Delta_\alpha + \Delta_\beta; \alpha_1; t_{\alpha_1} := t)^{\alpha\beta}]G_\alpha^{\alpha\beta}}}{\mathcal{E}, A_\alpha, A_\beta, G_\alpha^{\alpha\beta} \wedge (t_{\beta_1} = t \to G_\beta)^{\alpha\beta} \vdash [(?t_{\beta_1} < t \leqslant t_{\alpha_1} + \Delta_\alpha + \Delta_\beta; \alpha_1; t_{\alpha_1} := t)^{\alpha\beta}](G_\alpha^{\alpha\beta} \wedge (t_{\beta_1} = t \to G_\beta)^{\alpha\beta})} \text{[BoxAnd]}}{\mathcal{E}, A_\alpha, A_\beta, G_\alpha^{\alpha\beta} \wedge (t_{\beta_1} = t \to G_\beta)^{\alpha\beta} \vdash [body_\alpha^{\alpha\beta}](G_\alpha^{\alpha\beta} \wedge (t_{\beta_1} = t \to G_\beta)^{\alpha\beta})} \text{Unfolding}$$

It remains to derive a proof tree for the sequent $\mathcal{E}, A_\alpha, A_\beta, G_\alpha^{\alpha\beta} \wedge (t_{\beta_1} = t \to G_\beta)^{\alpha\beta} \vdash [(?t_{\beta_1} < t \leqslant t_{\alpha_1} + \Delta_\alpha + \Delta_\beta; \alpha_1; t_{\alpha_1} := t)^{\alpha\beta}](t_{\beta_1} = t \to G_\beta)^{\alpha\beta}$. We first weaken the left side of our sequent with the rule W_l because these formulas will not be needed. We symbolically execute $(?t_{\beta_1} < t \leqslant t_{\alpha_1} + \Delta_\alpha + \Delta_\beta; \alpha_1; t_{\alpha_1} := t)^{\alpha\beta}$ which yields in the hypothesis the formulas $t_{\beta_1} \lneqq t$ and $t \leqslant t_{\alpha_1} + \Delta_\alpha + \Delta_\beta$.

We apply the rule $\to_r$ and have two contradictory formulas in the left-side of the sequent. We can thus apply the *ex falso quod libet* (rule $\perp$) to conclude.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{t_{\beta_1} < t, t \leqslant t_{\alpha_1} + \Delta_\alpha + \Delta_\beta, t_{\beta_1} = t \vdash G_\beta} \perp}{t_{\beta_1} < t, t \leqslant t_{\alpha_1} + \Delta_\alpha + \Delta_\beta \vdash t_{\beta_1} = t \to G_\beta^{\alpha\beta}} \to_r}{\vdash [(?t_{\beta_1} < t \leqslant t_{\alpha_1} + \Delta_\alpha + \Delta_\beta; \alpha_1; t_{\alpha_1} := t)^{\alpha\beta}](t_{\beta_1} = t \to G_\beta)^{\alpha\beta}} \text{Symbolic execution}}{\mathcal{E}, A_\alpha, A_\beta, G_\alpha^{\alpha\beta} \wedge (t_{\beta_1} = t \to G_\beta)^{\alpha\beta} \vdash [(?t_{\beta_1} < t \leqslant t_{\alpha_1} + \Delta_\alpha + \Delta_\beta; \alpha_1; t_{\alpha_1} := t)^{\alpha\beta}](t_{\beta_1} = t \to G_\beta)^{\alpha\beta}} W_l$$

□

Conclusion

We have presented how to order discrete components using notifications introduced in Section 4.2. For a causal composition of a component `A` with a component `B`, we relax the condition stating that guarantees of a component `B` must not refer to outputs of a component `A`.

Chapter 5

Future works

In this chapter, we present research directions following our work. There is two categories of directions; one where we extend the expressiveness of our framework to take into account more systems, and one where we validate our approach against use cases. For each direction discussed, it is yet unclear if the results hold or whether it will prove to be useful.

We organize them in three sections. Section 5.1 presents research ideas continuing the results of Chapter 3. The Section 5.2 presents research ideas continuing the results of Chapter 4. The last Section 5.3 presents possible ideas on integrating refinement into component-based approach.

5.1 Continuing from the results on parallel composition

There are three main directions to investigate from the results of Chapter 3. Recall that we have defined a parallel composition operator to modularly model cyber-physical systems and a composition theorem ensuring that the component resulting from a parallel composition of two components satisfies the conjunction of respective contracts, allowing a modular proof of correctness of the system. We have also described a prototype to implement the process of construction of a proof of satisfaction of respective contracts.

First, we may want to modify our continuous composition operator $\circ_c$ to take into account a wider variety of models. We discuss possible ideas and difficulties that can be encountered. Secondly, we think that it is possible to relax the condition of Theorem 4 requiring that the guarantee of a component must not refer to outputs of an other component. Last, we discuss improvement points of the implementation of our prototype.

5.1.1 Extending the parallel continuous composition with complementary domains

When composing in parallel two continuous components, we define the resulting behavior, a differential equation, on the conjunction of evolution domains. The main reason for this choice is that we consider systems with distinct outputs. We also require that the evolution domain of a continuous behavior must not refer to outputs of other components. These requirements are crucial to ensure that the resulting component satisfies the conjunction of contracts. Given these conditions, two evolution domains should not interfere, making the choice of considering the conjunction of evolution domains not too restrictive.

Given two continuous components, the parallel composition of their behaviors is defined to be the system of both differential equations on the conjunction of evolution domains. We recall the definition of parallel continuous composition. Let $\alpha \triangleq \dot{X} = \theta_X \ \& \ H_X$ and $\beta \triangleq \dot{Y} = \theta_Y \ \& \ H_Y$ be continuous behaviors of components `A` and `B`. The parallel composition is:

$$\alpha \circ_c \beta \triangleq \dot{X} = \Theta_X, \dot{Y} = \Theta_Y \ \& \ H_X \land H_Y$$

There exist systems that we would like to compose in parallel, but for which this operator is not fitted. For example, consider two water-levels, one running during the duration $[0, 5]$, the other one running during the duration $[3, 7]$. We want to model the system during the duration $[0, 7]$. Yet, if we apply the parallel composition operator, we obtain the execution only on the duration $[3, 5]$.

Example 62 (Two water-levels)**.** *Let $\dot{wl_1} = fin_1 - fout_1, \dot{t} = 1 \ \& \ 0 \leqslant t \leqslant 5$ and $\dot{wl_2} = fin_2 - fout_2, \dot{t} = 1 \ \& \ 3 \leqslant t \leqslant 7$ be two behaviors.*

Their parallel composition results in the following differential equation:

$$\dot{wl_1} = fin_1 - fout_1, \dot{wl_2} = fin_2 - fout_2, \dot{t} = 1 \;\; \& \;\; 0 \leqslant t \leqslant 5 \wedge 3 \leqslant t \leqslant 7$$

which is equivalent to

$$\dot{wl_1} = fin_1 - fout_1, \dot{wl_2} = fin_2 - fout_2, \dot{t} = 1 \;\; \& \;\; 3 \leqslant t \leqslant 5$$

We lose the evolution of the first water-level on the duration $[0, 3]$ *and the evolution of the second water-level on the duration* $[5, 7]$.

There is one global variable that is read by every component: the time. A restriction may arise from time-related properties as in Example 62. In our work, we are mainly interested in systems that run for an arbitrary long period of time, and this kind of time-related properties is unlikely to arise. Yet, it may be interesting to investigate how to modularly model this kind of systems.

Extension with complementary domains A possible adaptation is to consider the evolution on complementary domains as in the work of Ronkko *et al.* [114]. The continuous parallel composition of two differential equations $\dot{X} = \theta_X \;\; \& \;\; H_X$ and $\dot{Y} = \theta_Y \;\; \& \;\; H_Y$ would be given by

$$\big((\dot{X} = \theta_X, \dot{Y} = \theta_Y \;\; \& \;\; H_X \wedge H_Y) \cup (\dot{X} = \theta_X \;\; \& \;\; H_X \wedge \overline{H_Y}) \cup (\dot{Y} = \theta_Y \;\; \& \;\; \overline{H_X} \wedge H_Y)\big)^*$$

We have three differential equations separated with a non-deterministic choice. The notation $\overline{H_X}$ means the complement of the domain defined by H_X. We iterate the whole to allows to pass freely from a differential equation to another one during the execution. The proposed definition still matches our general form of behavior.

Example 63 (Two water-levels on complementary domains)**.** *Applying it to the two water-level example results in the following model:*

$$\left(\begin{array}{ll} & \dot{wl_1} = fin_1 - fout_1, \dot{t} = 1 \;\; \& \;\; 0 \leqslant t < 3 \\ \cup & \dot{wl_1} = fin_1 - fout_1, \dot{wl_2} = fin_2 - fout_2, \dot{t} = 1 \;\; \& \;\; 3 \leqslant t \leqslant 5 \\ \cup & \dot{wl_2} = fin_2 - fout_2, \dot{t} = 1 \;\; \& \;\; 5 < t \leqslant 7 \end{array}\right)^*$$

On the duration $[0, 3)$*, we consider only the execution of the first water-level, and reciprocally we consider only the execution of the second water-level in the duration* $(5, 7]$*. For the duration* $[3, 5]$*, we have both executions in parallel. We pass from one equation to the other with the passing of time.*

There are several points to investigate. First, we have chosen to not specify the evolution of the differential equation $\dot{X} = \theta_X$ on the evolution domain $\overline{H_X}$. An other option, the *linear continuous parallel composition* in the work of Ronkko *et al.* [114], is to specify that $\dot{X} = 0$ on the complementary domain $\overline{H_X}$. It means that we assume that the system does not evolve when it is not on its evolution domain.

Although the linear parallel composition is more restrictive, it also has more interesting properties. According to Ronkko *et al.*, it is associative when the case where the evolution is not specified does not enjoy associativity. We think that this result may extend to the

definition presented in this subsection. Such property would have to be investigated to determine whether this extension of the continuous parallel composition is useful to our modular component-based approach.

A very important point of our approach is the guarantee that the component resulting from the parallel composition of two components satisfies the conjunction of respective contracts. It remains to investigate whether such result holds for both versions of the extension of the parallel continuous composition operator with complementary domains.

Lastly, we have to be cautious is that this extension of continuous parallel composition does not build models with a combinatorial size, obliterating the scalability of such system. Considering complementary domains may lead to such problem ad it may be possible to have to consider methods to tame it.

5.1.2 Relaxing conditions of composition theorem

One of the necessary conditions to retain contracts through parallel composition is that the guarantees of a component must not refer to the outputs of an other component. It is also a good design practice. Yet, we may want express a guarantee according to outputs of other component. For example, the water-level controller has the guarantee $wlm \leqslant 3.5 \rightarrow fin = 1$ which means that if the water-level measured wlm is below 3.5, we have to open the inlet valve ($fin = 1$). Assume we split the controller in two sub-components, a sensor to measure the value and a monitor deciding the course of action according to the value of the sensor. Then the guarantee $wlm \leqslant 3.5 \rightarrow fin = 1$ is the one of the monitor, but it refers to an output of another component, the sensor.

But we think that it is possible to relax this condition. Assume that the guarantee G_α of a component `A` is of the form $p \rightarrow \varphi_\alpha$ where p is a literal and φ_α is a formula. Instead of requiring that there is no occurrences of outputs of another component `B` in $p \rightarrow \varphi_\alpha$, we forbid such occurrences only in φ_α. We can refer to outputs of `B` in p. The idea behind this relaxation is that p is a "hidden hypothesis". We require thus that p is implied by the guarantees of `A`.

Conjuncture 1. *Let α and β be two behaviors of general form of components `A` and `B` with respective contracts (A_α, G_α) and (A_β, G_β). Assume that G_α is of the form $p_\alpha \rightarrow \varphi_\alpha$ and G_β of the form $p_\beta \rightarrow \varphi_\beta$. Assume that we have two proof trees of $\mathcal{E}, A_\alpha \vdash [\alpha]G_\alpha$ and $\mathcal{E}, A_\beta \vdash [\beta]G_\beta$ respectively, where $\mathcal{E}$ is the environment. Furthermore, assume that*

- (a) $BV(\alpha) \cap BV(\beta) = \varnothing$,
- (b_1) $BV(\alpha) \cap FV(\varphi_\beta) = \varnothing$,
- (b_2) $BV(\beta) \cap FV(\varphi_\alpha) = \varnothing$,
- (c_1) $\mathcal{E}, A_\alpha \vdash \forall^\beta\big(G_\beta \rightarrow (A_\alpha \wedge p_\alpha)\big)$,
- (c_2) $\mathcal{E}, A_\beta \vdash \forall^\alpha\big(G_\alpha \rightarrow (A_\beta \wedge p_\beta)\big)$.

Then it exists a proof tree of $\mathcal{E}, A_\alpha, A_\beta \vdash [\alpha \circ_c \beta](G_\alpha \wedge G_\beta)$.

Proof sketch We provide a proof sketch of the conjuncture. Recall that the behavior α is of the form $(\mathbf{disc}_\alpha \cup \mathbf{cont}_\alpha)^*$ and β is of the form $(\mathbf{disc}_\beta \cup \mathbf{cont}_\beta)$. The parallel composition $\alpha \circ \beta$ results in the behavior $\big(\mathbf{disc}_\alpha \cup \mathbf{disc}_\beta \cup (\mathbf{cont}_\alpha \circ_c \mathbf{cont}_\beta)\big)^*$.

The idea of the proof is to use the conjunction of guarantees $G_\alpha \wedge G_\beta$ as an invariant for the induction rule. We separate $\mathbf{disc}_\alpha$, $\mathbf{disc}_\beta$ and $\mathbf{cont}_\alpha \circ_c \mathbf{cont}_\beta$ with the non-deterministic choice rule $[\cup]$. We have to prove the sequent $\mathcal{E}, A_\alpha, A_\beta, G_\alpha^{\alpha\beta}, G_\beta^{\alpha\beta} \vdash [\mathbf{disc}_\alpha^{\alpha\beta}](G_\alpha^{\alpha\beta} \wedge G_\beta^{\alpha\beta})$. We have similar sequents for the branches corresponding to $\mathbf{disc}_\beta$ and $\mathbf{cont}_\alpha \circ_c \mathbf{cont}_\beta$. We split the formula under the modality with the [BoxAnd] rule to distinct the cases of $G_\alpha^{\alpha\beta}$ and $G_\beta^{\alpha\beta}$.

To prove the sequent $\mathcal{E}, A_\alpha, A_\beta, G_\alpha^{\alpha\beta}, G_\beta^{\alpha\beta} \vdash [\mathbf{disc}_\alpha^{\alpha\beta}]G_\beta^{\alpha\beta}$, we use the separation lemma in the proof of Theorem 4 and the fact that G_β does not refer to bound variables of $\mathbf{disc}_\alpha$, and thus is not modified by the execution of $\mathbf{disc}_\alpha$. Here, since G_β refers to bound variables of $\mathbf{disc}_\alpha$, we cannot apply this mechanism immediately, we have to be more cautious.

We first unfold the formula $G_\beta^{\alpha\beta}$. The step (i) is the application of the separation lemma (cf Lemma 1 to the formula $\varphi_\beta^{\alpha\beta}$. We then symbolically execute $\mathbf{disc}_\alpha$ which does not modify φ_β^β, but may modify $p_\beta^{\alpha\beta}$. To mark this possible change, we use the superscript $^{\mathbf{disc}_\alpha}$. A consequence is that $p_\beta^{\alpha\beta}$ and $(p_\beta^{\alpha\beta})^{\mathbf{disc}_\alpha}$ are possibly different formulas, and we can not use them to conclude. Remember that we have the condition (c_2) which allow us to obtain the formula p_β from G_α, and thus $p_\beta^{\alpha\beta}$ from $G_\alpha^{\alpha\beta}$. The conclusion is straightforward.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathcal{E}, A_\alpha, A_\beta, G_\alpha^{\alpha\beta}, p_\beta^{\alpha\beta} \to \varphi_\beta^\beta, (p_\beta^{\alpha\beta})^{\mathbf{disc}_\alpha} \vdash \varphi_\beta^\beta}}{\mathcal{E}, A_\alpha, A_\beta, G_\alpha^{\alpha\beta}, p_\beta^{\alpha\beta} \to \varphi_\beta^\beta \vdash (p_\beta^{\alpha\beta})^{\mathbf{disc}_\alpha} \to \varphi_\beta^\beta} \to_r}{\mathcal{E}, A_\alpha, A_\beta, G_\alpha^{\alpha\beta}, p_\beta^{\alpha\beta} \to \varphi_\beta^\beta \vdash [\mathbf{disc}_\alpha^{\alpha\beta}]p_\beta^{\alpha\beta} \to \varphi_\beta^\beta} \text{Symbolic execution of } \mathbf{disc}_\alpha}{\mathcal{E}, A_\alpha, A_\beta, G_\alpha^{\alpha\beta}, p_\beta^{\alpha\beta} \to \varphi_\beta^{\alpha\beta} \vdash [\mathbf{disc}_\alpha^{\alpha\beta}]p_\beta^{\alpha\beta} \to \varphi_\beta^{\alpha\beta}} (i)}{\mathcal{E}, A_\alpha, A_\beta, G_\alpha^{\alpha\beta}, G_\beta^{\alpha\beta} \vdash [\mathbf{disc}_\alpha^{\alpha\beta}]G_\beta^{\alpha\beta}} \text{Unfolding } G_\beta^{\alpha\beta}$$

The technical reason is that the literal p is in negative position. It is thus possible to extend it to the more general condition: "Occurrences of outputs of the component `B` can only occurs into negative formulas".

5.1.3 Addition of communication channels

Our model of communication is based on shared variables. It is a simplistic model and is not realistic to tackle more complex systems. We present in this subsection how we can define communicating channels in a similar manner to the primitives of HCSP. We present how it can be used to revisit our composition theorem 4. We also outline how it will be possible to integrate communication delays.

Definition of communicating channels We follow the encoding proposed by Guelev *et al.* [56] to model inputs and outputs with several dedicated shared variables.

$ch!e$ means that we send the value e along the channel ch and $ch?x$ means that we assign the value received on the channel ch to the variable x. A very naive way to encode this is by $ch!e \triangleq ch := e$ and $ch?x \triangleq x := ch$ which is just variable sharing with one extra step. In such

setting, the sender does not care if the value has been received. It may send several times a value (*e.g.* a sensor which repeatedly measure some physical variable sent to a controller) when the receiver only receive one of them (*e.g.* , the controller use one of two updates). It is a very loose specification.

A less naive encoding is given by:

$$\begin{aligned} ch!e &\triangleq ?rcv = 1; ch := e; rcv := 0 \\ ch?x &\triangleq ?rcv = 0; x := ch; rcv := 1 \end{aligned}$$

where rcv is a dedicated variable accounting to know if the value has been received or not. With this encoding a sensor can not send a new value unless the controller has acknowledged to have receive one.

Yet, it is not the way communications are modeled in HCSP. In Hybrid Communicating Sequential Processes, a communication channel is blocking, *i.e.* that a process can not continue if the reception has not been acknowledged. For example, in the program $ch!e; \alpha$, α can not execute until e has been received by the other side of the channel. The proposed encoding is the following one:

$$\begin{aligned} ch!e &\triangleq ch := e; ch! := \top; \textbf{await } ch?; \textbf{await } \neg ch?; ch! := \bot \\ ch?e &\triangleq ch? := \top; \textbf{await } ch!; x := ch; ch? := \bot; \textbf{await } \neg ch! \end{aligned}$$

where $ch?$ and $ch!$ are dedicated boolean variables. **await** $ch?$ means that we wait for the variable $ch?$ to be true, *i.e.* that the receiver is ready. Then, we wait that the receiver signals that it has effectively received the value ($\neg ch?$ is true). For the receiver, the encoding is a mirror.

Alas, this encoding is not directly usable in $d\mathcal{L}$. Indeed, it uses the fact that parallelism is a base feature of the formalism. If we directly use this definition, we would never receive the acknowledgment sent by the receiver since we do not let a possibility to the receiver to executes. We need to adapt the encoding to comply with our notion of parallelism based on the non-deterministic choice operator.

The idea is to separate the encoding of $ch!e$ and $ch?x$ of α by the non-deterministic choice operator $\cup$ to retrieve our framework. We have to add ordering constraints to ensure that $ch!e$ executes before α. We make extensive use of guards in a similar manner to the encoding of Section 4.2. We proposed the following encoding:

$$\begin{aligned} ch!e &\triangleq (?ch? = \top \wedge ch! = \bot; ch := e; ch! := \top) \\ &\quad \cup (?(ch? = \bot \wedge ch! = \top) \vee (ch? = \top \wedge ch! = \top); ch! := \bot) \\ ch?x &\triangleq (?ch! = \top \wedge ch? = \top; x := ch; ch? := \bot) \\ &\quad \cup (?((ch? = \bot \wedge ch! = \top) \vee (ch? = \bot \wedge ch! = \bot) \wedge x = ch; ch? := \top \end{aligned}$$

We also use two dedicated boolean variables $ch!$ and $ch?$ to perform the ordering constraints. The two constructs in the right-hand side of the non-deterministic choice operator $\cup$ are here to account for the waiting periods **await** $\neg ch?$ and **await** $ch!$. Using the non-deterministic choice operator allows time to flow between two executions.

We show how it interacts with other constructs. Let α and β be discrete programs. We define $ch!e; \alpha$ and $ch?x; \beta$ as following:

$$
\begin{array}{lll}
ch!e;\alpha & \triangleq & (?ch? = \top \wedge ch! = \bot; ch := e; ch! := \top) \\
 & & \cup(?(ch? = \bot \wedge ch! = \top) \vee (ch? = \top \wedge ch! = \top); \alpha; ch! := \bot) \\
ch?x;\beta & \triangleq & (?ch! = \top \wedge ch? = \top; x := ch; ch? := \bot) \\
 & & \cup(?((ch? = \bot \wedge ch! = \top) \vee (ch? = \bot \wedge ch! = \bot)) \wedge x = ch; \beta; ch? := \top
\end{array}
$$

We do not use the sequence operator ; as it does not allow time to flow between communications. But the proposed encoding is equivalent in terms of reachability semantic. Remember that we assume our components to always be iterated an arbitrary number of times. Thus we compare the reachability set of $(ch!e;\alpha)^*$ and $\big((?ch? = \top \wedge ch! = \bot; ch := e; ch! := \top) \cup (?ch? = \bot \wedge ch! = \top; \alpha; ch! := \bot)\big)^*$.

Conjuncture 2.

$$
\rho\Big(\big(ch!e;\alpha\big)^*\Big) = \rho\Big(\big((?ch? = \top \wedge ch! = \bot; ch := e; ch! := \top) \cup (?ch? = \bot \wedge ch! = \top; \alpha; ch! := \bot)\big)^*\Big)
$$

We can use the same idea to encode the other usual interactions such as non-deterministic choice and iteration.

Revisiting the parallel composition Since we have an explicit notion of communication channels, we can assume that they are unique and more subtle during the composition for the resulting inputs and outputs. When composing two components with matching inputs and outputs, we can hide it at the level of the component resulting from the parallel composition. The communication channel is internalized and communicates no more with the exterior. The Figure 5.1 details the process.

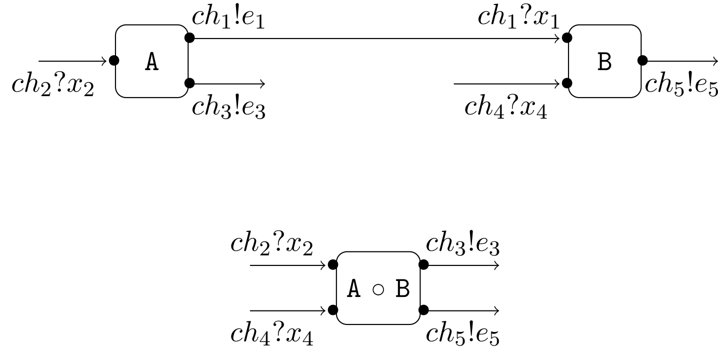


Figure 5.1: Parallel composition with communication channels

In the Figure 5.1, the channel ch_1 is hidden by the composition. It helps the scalability of our method since it allows to hide variables that are of no use during after the composition. Such process is not possible using our simple model of communications based on shared variables. Indeed, an output may be used by several components and hiding it during the composition may forbid associative composition and incremental design.

Communication delays Communication delays are a serious issue for the design of hybrid systems in a compositional manner. The addition of new components may introduce

delays that were not expected and cause malfunction. We show how to add delays to our communication channels using the work of Section 4.2.

Assume that the sending of a message takes Δ_1 units of time and the acknowledgment Δ_2 units of time. We add two special notification variables $t_{ch!}$ and $t_{ch?}$ to remember the last execution. We add the tests $?t \leqslant t_{ch!} + \Delta_1$ and $?t \leqslant t_{ch?} + \Delta_2$ in the guards and the assignment $t_{ch!} := t$ and $t_{ch?} := t$ as following:

$$\begin{aligned}
ch!e \quad &\triangleq \quad (?ch? = \top \wedge ch! = \bot \wedge t \leqslant t_{ch!} + \Delta_1; ch := e; ch! := \top; t_{ch!} := t) \\
&\quad\ \cup(?(ch? = \bot \wedge ch! = \top) \vee (ch? = \top \wedge ch! = \top); ch! := \bot) \\
ch?x \quad &\triangle \quad q(?ch! = \top \wedge ch? = \top \wedge t \leqslant t_{ch?} + \Delta_2; x := ch; ch? := \bot; t_{ch?} := t) \\
&\quad\ \cup(?((ch? = \bot \wedge ch! = \top) \vee (ch? = \bot \wedge ch! = \bot)) \wedge x = ch; ch? := \top
\end{aligned}$$

We detail the case of $ch!e; \alpha$ where α (resp. β) is a discrete program and has an execution period of δ_α (resp. Δ_β).

$$\begin{aligned}
ch!e \quad &\triangleq \quad (?ch? = \top \wedge ch! = \bot \wedge t \leqslant t_{ch!} + \Delta_1; ch := e; ch! := \top; t_{ch!} := t) \\
&\quad\ \cup(?((ch? = \bot \wedge ch! = \top) \vee (ch? = \top \wedge ch! = \top)) \wedge t \leqslant t_\alpha + \Delta_\alpha; \alpha; t_\alpha := t; ch! := \bot) \\
ch?x \quad &\triangleq \quad (?ch! = \top \wedge ch? = \top \wedge t \leqslant t_{ch?} + \Delta_2; x := ch; ch? := \bot; t_{ch?} := t) \\
&\quad\ \cup(?((ch? = \bot \wedge ch! = \top) \vee (ch? = \bot \wedge ch! = \bot)) \wedge x = ch \wedge t \leqslant t_\beta + \Delta_\beta; \beta; t_\beta := t; ch? := \top
\end{aligned}$$

Our composition operator can be modified accordingly to take into account communication delays.

Conclusion We have presented ideas on the integration of communications channels in our framework, improving further the usability of our work. It allows the modeling of communication delays. It may also served as a basis to take into account privacy and security concerns. Indeed, being able to model a communication channel allows the modeling of attackers.

5.1.4 Implementation

We have presented in Section 3.4 a prototype implementing the process of construction of a proof of satisfaction of the conjunction of contracts in KeYmaera X. It shows feasibility of an automation of our approach, yet we are far from an usable tool. We discuss several directions to improve our prototype.

First, we need to take into account all the subtleties of the implementation of $d\mathcal{L}$ in KeYmaeraX. This theorem prover is in active development, and it is difficult to keep track of all introduced variations of proof rules.

A second is to implement the effect of the parallel composition operator. For now, each component are defined in the form of a .kyx file and the component resulting of the parallel composition is manually defined in a separate .kyx file. Since the parallel composition operator is syntactically defined, there should not be theoretical issues to this work.

With the two previous steps achieved, it will allow to test the effectiveness of our approach against more use-cases. It will help to define by practice for which kind of systems our approach is really effective and compare it with the component-based approach of Mueller *et al.* [88] and the standard approach in KeYmaera X.

5.2 Improving extensions of the parallel composition

We discuss research directions coming from results of Chapter 4. In this chapter, we have defined extensions of our parallel composition operator to take into accounts several category of systems. We have first defined a framework to take into account timing aspect inherent to Computer-Controlled Systems (CCS). We have also presented how to adapt the approach to model modes in a system and causality relations between discrete components.

First, we discuss in Subsection 5.2.1 how to handle timing aspects of CCS when there is more than one computation unit. In Subsection 5.2.2, we investigate the extension to continuous component our approach to handle modes in Cyber-Physical Systems. Last, we discuss also a possible extension of our causal composition operator to continuous component in Subsection 5.2.3.

5.2.1 Timed parallel composition with several CPUs

We have presented in Section 4.2 how to take into account execution periods during parallel composition. We are in the context of Computer-Controlled Systems where discrete components are in fact programs. For that, we have assumed that we execute the programs on one CPU, leading that the resulting execution period of two programs in parallel is the sum of respective execution periods. An interesting point to investigate is if we assume that each program have its own CPU. Thus the resulting execution period from the parallel composition would be the maximum of each resulting periods.

Given this slight change in the definition of a timed parallel composition operator (cf Definition 46), we should retain the commutativity and associativity properties. Also, we should still be able to guarantee that the component resulting from this parallel composition satisfies the conjunction of respective contracts, *i.e.* a theorem similar to Theorem 10.

Yet, it seems a little unrealistic to assume that every program is executed on its own CPU and it may be too restrictive to assume that they all execute on one CPU. An interesting point to investigate would be how we can define a timed parallel composition operator when we have two CPUs, three CPUs, or a fixed number n of CPUs. More generally, how it is possible to integrate results on WCET calculus in our framework.

5.2.2 Modes for continuous systems

We have defined modes only for discrete systems in Section 4.3. It is justified by the fact that modes are inherently human-made constructs, and thus it seems unnatural to define modes on continuous components. Yet, there may be systems outside our knowledge to which such modeling will be useful. It may also be useful as a proof artifact.

A possible modeling would be to add the events E in the evolution domain. To accounts for the two possible modes, we would have to split our continuous system in two differential equation and use the non-deterministic choice operator for consistency with our previous definition. It is somehow similar to the idea in Subsection 5.1.1.

Given two behaviors $\dot{X} = \theta_X \ \& \ H_X$ and $\dot{Y} = \theta_Y \ \& \ H_Y$, and a mode event E, the mode composition would result in the following behavior

$$\big((\dot{X} = \theta_X \ \& \ H_X \wedge E) \cup (\dot{Y} = \theta_Y \ \& \ H_Y \wedge \neg E)\big)^*$$

It is unclear how it would interact with discrete components, and whether it is associative. Also, it is not clear whether the same reasoning as in Subsection 4.3.2 can be applied to retain contracts through composition.

5.2.3 Causal composition for continuous components

As for the mode structure, we have defined the causal composition for discrete components. We think that an extension to continuous component is possible, but it is not clear if there is a need for it. However, it may be useful as a proof artifact.

Causal composition of two components means that one has to execute before the other one. The obvious choice to model it is to use the sequence operator ; of $d\mathcal{L}$. But, it does not fit our general form of a component's behavior with the use of non-deterministic choice. A possibility is to make profit of the control period notion to temporally separate them.

Definition 52 (Continuous causal composition)**.** *Let $\alpha \triangleq \dot{X} = \theta_X \;\&\; H_X \wedge t \leqslant \Delta_\alpha$ be the behavior of a component* `A` *with control period Δ_α and $\beta \triangleq \dot{Y} = \theta_Y \;\&\; H_Y \wedge t \leqslant \Delta_\beta$ be the behavior of a component* `B` *with control period Δ_β. The continuous causal composition of these behaviors would be:*

$$\big((\dot{X} = \theta_X \;\&\; H_X \wedge t \leqslant \Delta_\alpha) \cup (\dot{Y} = \theta_Y \;\&\; H_Y \wedge \Delta_\alpha \leqslant t \leqslant \Delta_\beta + \Delta_\alpha)\big)^*$$

We first execute α, then β. It is similar to the idea of continuous composition presented in Subsection 5.1.1. Yet, there is several issues with this definition. First, it is incompatible with the rest of the framework where the continuous part is only one differential equation. This seems more compatible if we have already extended the definition of the continuous parallel composition as discussed in Subsection 5.1.1. A second issue is how it would interact with the discrete part. Two last issues are whether this composition is associative and whether the component resulting from this composition would satisfies the conjunction of contracts.

5.3 Toward integration of refinement into component-based approach

5.3.1 Refinement in $d\mathcal{L}$

In [86], the authors define a notion of refinement for hybrid programs. A hybrid program α refines a hybrid program β if the set of reachable states of α is included in the set of reachable states of β. Recall that we note $\rho(\alpha)$ the set of reachable states of α.

Definition 53 (Refinement)**.** *A hybrid program α refines a hybrid program β, denoted by $\alpha \sqsubseteq \beta$, if*

$$\rho(\alpha) \subseteq \rho(\beta)$$

In [81], the authors integrate the notion of refinement as a first-class formula of the system and provide an extension of the sequent calculus to syntactically reason on refinement. The formula stating that α refines β is denoted by $\alpha \leqslant \beta$. Its semantic follows from the previous definition.

Definition 54 (Semantic $\alpha \leqslant \beta$)**.**

$$\nu \models \alpha \leqslant \beta \textit{ iff } \{\omega \mid (\nu, \omega) \in \rho(\alpha)\} \subseteq \{\omega \mid (\nu, \omega) \in \rho(\beta)\}$$

An important rule associated to this formula is the rule $[\leqslant]$. To prove that φ holds for all executions of α, it is sufficient to prove that φ holds for all executions of β and that α refines β.

$$\frac{\Gamma \vdash [\beta]\varphi \qquad \Gamma \vdash \alpha \leqslant \beta}{\Gamma \vdash [\alpha]\varphi}\,[\leqslant]$$

The rule is correct. Indeed, if φ holds for every executions of β, and that executions of α are just a subset of executions of β, φ holds also for all executions of α.

The authors provide also rules to derive the validity of formula $\alpha \leqslant \beta$ following the structure of α and β. For example, we have the following rule.

$$\frac{\Gamma \vdash \alpha_1 \leqslant \alpha_2 \qquad \Gamma \vdash [\alpha_1](\beta_1 \leqslant \beta_2)}{\Gamma \vdash (\alpha_1; \beta_1) \leqslant (\alpha_2; \beta_2)}\,(;)$$

To show that $\alpha_1; \beta_1$ refines $\alpha_2; \beta_2$, we have to show that α_1 refines α_2 and, after all runs of α_1, that β_1 also refines β_2.

5.3.2 Refinement and parallel composition

When designing an industrial system, it is common to update a component, *e.g.* a sensor is updated with a new model that samples twice faster. Yet, we do not want to redo all our design and proof only to u pdate one component. A very interesting feature would be to say that if the behavior of a component `B` refines the behavior of a component `A`, with same contract and input/outputs, then we can replace `A` by `B`. We think that our parallel component-based operator enjoys this property.

Conjuncture 3 (Update of a component)**.** *Let α, α' and β be behaviors of component. If $\alpha_1 \leqslant \alpha_2$, then :*

$$(\alpha_1 \otimes \beta) \leqslant (\alpha_2 \otimes \beta)$$

Conclusion

Our initial goal was how to modularly design and prove correctness of Cyber-Physical Systems (CPS)? Such systems feature physical evolution with discrete interactions. Such systems are pervasive, but their dual nature make them hard to model and verify. We are more specifically interested into what we call *structural complexity*. It refers to systems where blocks are elementary and the complexity of the system arise from their repetitive composition. The challenge is to come up with a correct-by-design methodology usable by an engineer. For that, it must follow as much as possible the usual workflow of designer of Cyber-Physical Systems while integrating proof methods to ensure correctness of the system.

To tackle the structural complexity, we have developed a modular component-based approach in the Differential Dynamic Logic ($d\mathcal{L}$) presented in the Chapter 3. We have shown also how to adapt it to several common design patterns in the Chapter 4.

More precisely, we have defined a notion of component at a high-level specification, the so-called textual representation, and at a low-level specification, the so-called behavior, as a hybrid program of $d\mathcal{L}$. We have defined what it means for a component to satisfy its contract and exemplified it with a cruise-control example.

We have defined how to compose parallel components by mean of a parallel composition operator. We have shown that it is commutative, meaning that the order of composition is not important, and associative, meaning that we can incrementally design a system. We hence have a truly modular design method for CPS.

On top of that, we have stated and demonstrated a theorem that allows to construct contracts through composition. Given two components that satisfy their respective contracts, the component resulting from the parallel composition satisfies the conjunction of contracts. The theorem operates under conditions that correspond to required properties for a meaningful composition. It has also been exemplified with the cruise-control example.

To assess the feasibility of a possible implementation of our method, we have developed a small prototype using the tactic language defined in KeYmaera X. Although it is just a working prototype, it comforts us to think that a full-fledged implementation is possible.

Finally, to explore the possibility of our methodology, we have studied a small water-plant example where two water-tanks are linked together and they have their water-level controller. It shows that our methodology is general, and requires a lot of supplementary efforts from the designer to conceive Computer-Controlled Systems, a special class of Cyber-Physical Systems widely used in industrial systems.

To remedy such facts, we have studied several adaptations of our parallel composition operator to take into account the timing constraints on executions of a system, the introduction of modes and the causal composition of two components.

For each adaptation, we have presented how to modularly model and prove correctness of considered systems. We have be careful to retain the associativity property and that the adapted composition remains syntactic. We have also stated and demonstrated for each case a theorem ensuring that the component resulting from the composition satisfies the conjunction of respective contracts.

We have presented a modular framework to model and prove correctness of Cyber-Physical Systems. We have showed it was adaptable if necessary. Yet, we have to validate it against more complex use cases to determine its effectiveness. For that, a complete implementation will be necessary. Validation against use cases will also help to determine if there is a need for more extensions of our framework or if the presented approach is sufficient.

Bibliography

[1] Proof measurement and estimation. `https://ts.data61.csiro.au/projects/TS/pme.pml`.

[2] Alessandro Abate, Maria Prandini, John Lygeros, and Shankar Sastry. Probabilistic reachability and safety for controlled discrete time stochastic hybrid systems. *Automatica*, 44(11):2724–2734, 2008.

[3] Jean-Raymond Abrial. *The B-book: assigning programs to meanings*. Cambridge University Press, 2005.

[4] Jean-Raymond Abrial, Wen Su, and Huibiao Zhu. Formalizing hybrid systems with event-b. In *International Conference on Abstract State Machines, Alloy, B, VDM, and Z*, pages 178–193. Springer, 2012.

[5] Luca Aceto, Augusto Burgueno, and Kim G Larsen. Model checking via reachability testing for timed automata. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 263–280. Springer, 1998.

[6] Wolfgang Ahrendt, Bernhard Beckert, Daniel Bruns, Richard Bubel, Christoph Gladisch, Sarah Grebing, Reiner Hähnle, Martin Hentschel, Mihai Herda, Vladimir Klebanov, et al. The key platform for verification and analysis of java programs. In *Working Conference on Verified Software: Theories, Tools, and Experiments*, pages 55–71. Springer, 2014.

[7] Eitan Altman and Vladimir Gaitsgory. Asymptotic optimization of a nonlinear hybrid system governed by a markov decision process. *SIAM Journal on Control and Optimization*, 35(6):2070–2085, 1997.

[8] Rajeev Alur, Costas Courcoubetis, Thomas A Henzinger, and Pei-Hsin Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In *Hybrid systems*, pages 209–229. Springer, 1993.

[9] Rajeev Alur, Radu Grosu, Insup Lee, and Oleg Sokolsky. Compositional modeling and refinement for hierarchical hybrid systems. *The Journal of Logic and Algebraic Programming*, 68(1-2):105–128, 2006.

[10] Rajeev Alur, Thomas A Henzinger, Gerardo Lafferriere, and George J Pappas. Discrete abstractions of hybrid systems. *Proceedings of the IEEE*, 88(7):971–984, 2000.

[11] Karl J Åström and Björn Wittenmark. *Computer-controlled systems: theory and design*. Courier Corporation, 2013.

[12] Ralph-Johan Back, Luigia Petre, and Ivan Porres. Continuous action systems as a model for hybrid systems. *Nord. J. Comput.*, 8(1):2–21, 2001.

[13] Ralph-Johan Back and Joakim Wright. *Refinement calculus: a systematic introduction.* Springer Science & Business Media, 1998.

[14] RJR Back and Joakim von Wright. *Trace refinement of action systems.* Springer, 1994.

[15] Jos CM Baeten. A brief history of process algebra. *Theoretical Computer Science*, 335(2-3):131–146, 2005.

[16] Richard Banach, Michael Butler, Shengchao Qin, Nitika Verma, and Huibiao Zhu. Core hybrid event-b i: single hybrid event-b machines. *Science of Computer Programming*, 105:92–123, 2015.

[17] Bernhard Beckert, Reiner Hähnle, and Peter H Schmitt. *Verification of object-oriented software: The KeY approach.* Springer-Verlag, 2007.

[18] Albert Benveniste, Benoît Caillaud, Dejan Nickovic, Roberto Passerone, Jean-Baptiste Raclet, Philipp Reinkemeier, Alberto Sangiovanni-Vincentelli, Werner Damm, Thomas Henzinger, and Kim Guldstrand Larsen. Contracts for system design. Technical report, 2012.

[19] Albert Benveniste, Paul Caspi, Stephen A Edwards, Nicolas Halbwachs, Paul Le Guernic, and Robert De Simone. The synchronous languages 12 years later. *Proceedings of the IEEE*, 91(1):64–83, 2003.

[20] Albert Benveniste, Dejan Nickovic, and Thomas Henzinger. Compositional Contract Abstraction for System Design. Research report, INRIA, 2014.

[21] Gérard Berry and Georges Gonthier. The esterel synchronous programming language: Design, semantics, implementation. *Science of computer programming*, 19(2):87–152, 1992.

[22] Yves Bertot and Pierre Castéran. *Interactive theorem proving and program development: Coq'Art: the calculus of inductive constructions.* Springer Science & Business Media, 2013.

[23] François Bobot, Jean-Christophe Filliâtre, Claude Marché, and Andrei Paskevich. Why3: Shepherd your herd of provers. In *Boogie 2011: First International Workshop on Intermediate Verification Languages*, pages 53–64, 2011.

[24] Brandon Bohrer and André Platzer. A hybrid, dynamic logic for hybrid-dynamic information flow. Technical report, Technical Report CMU-CS-18-105. School of Computer Science, Carnegie Mellon ..., 2018.

[25] Brandon Bohrer, Vincent Rahli, Ivana Vukotic, Marcus Völp, and André Platzer. Formally verified differential dynamic logic. In *Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs*, pages 208–221. ACM, 2017.

[26] Timothy Bourke and Marc Pouzet. Zélus: A synchronous language with odes. In *Proceedings of the 16th international conference on Hybrid systems: computation and control*, pages 113–118. ACM, 2013.

[27] Ed Brinksma, Tomas Krilavičius, and Yaroslav S Usenko. Process algebraic approach to hybrid systems. *IFAC Proceedings Volumes*, 38(1):325–330, 2005.

[28] Hendrik Brinksma and Tomas Krilavicius. Behavioural hybrid process calculus. 2005.

[29] Manuela L Bujorianu. Extended stochastic hybrid systems and their reachability problem. In *International Workshop on Hybrid Systems: Computation and Control*, pages 234–249. Springer, 2004.

[30] Manuela L Bujorianu, John Lygeros, and Marius C Bujorianu. Bisimulation for general stochastic hybrid systems. In *International Workshop on Hybrid Systems: Computation and Control*, pages 198–214. Springer, 2005.

[31] Zhou Chaochen, Charles Anthony Richard Hoare, and Anders P Ravn. A calculus of durations. *Information processing letters*, 40(5):269–276, 1991.

[32] Zhou Chaochen, Wang Ji, and Anders P Ravn. A formal description of hybrid systems. In *International Hybrid Systems Workshop*, pages 511–530. Springer, 1995.

[33] Mingshuai Chen, Xiao Han, Tao Tang, Shuling Wang, Mengfei Yang, Naijun Zhan, Hengjun Zhao, and Liang Zou. Mars: A toolchain for modelling, analysis and verification of hybrid systems. In *Provably Correct Systems*, pages 39–58. Springer, 2017.

[34] Edmund M Clarke, Orna Grumberg, and Kiyoharu Hamaguchi. Another look at ltl model checking. *Formal Methods in System Design*, 10(1):47–71, 1997.

[35] Edmund M Clarke, Orna Grumberg, and Doron Peled. *Model checking*. MIT press, 1999.

[36] Thierry Coquand and Gérard Huet. The calculus of constructions. *Information and computation*, 76(2-3):95–120, 1988.

[37] Patrick Cousot and Radhia Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 238–252. ACM, 1977.

[38] Patrick Cousot, Radhia Cousot, Jérôme Feret, Laurent Mauborgne, Antoine Miné, David Monniaux, and Xavier Rival. The astrée analyzer. In *European Symposium on Programming*, pages 21–30. Springer, 2005.

[39] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.

[40] Alexandre Donzé and Oded Maler. Systematic simulation using sensitivity analysis. In *International Workshop on Hybrid Systems: Computation and Control*, pages 174–189. Springer, 2007.

[41] Iulia Dragomir, Viorel Preoteasa, and Stavros Tripakis. Compositional semantics and analysis of hierarchical block diagrams. In *International Symposium on Model Checking Software*, pages 38–56. Springer, 2016.

[42] Ansgar Fehnker and Franjo Ivančić. Benchmarks for hybrid systems verification. In *Hybrid Systems: Computation and Control*, pages 326–341. Springer, 2004.

[43] Jean-Christophe Filliâtre and Andrei Paskevich. Why3—where programs meet provers. In *European Symposium on Programming*, pages 125–128. Springer, 2013.

[44] Martin Fränzle, Ernst Moritz Hahn, Holger Hermanns, Nicolás Wolovick, and Lijun Zhang. Measurability and safety verification for stochastic hybrid systems. In *Proceedings of the 14th international conference on Hybrid systems: computation and control*, pages 43–52. ACM, 2011.

[45] Goran Frehse. Phaver: Algorithmic verification of hybrid systems past hytech. In *International workshop on hybrid systems: computation and control*, pages 258–273. Springer, 2005.

[46] Nathan Fulton, Stefan Mitsch, Brandon Bohrer, and André Platzer. Bellerophon: Tactical theorem proving for hybrid systems. In *International Conference on Interactive Theorem Proving*, pages 207–224. Springer, 2017.

[47] Nathan Fulton, Stefan Mitsch, Jan-David Quesel, Marcus Völp, and André Platzer. Keymaerax: An axiomatic tactical theorem prover for hybrid systems. In *International Conference on Automated Deduction*, pages 527–538. Springer, 2015.

[48] Nathan Fulton and André Platzer. A logic of proofs for differential dynamic logic. 2016.

[49] Sicun Gao, Jeremy Avigad, and Edmund M Clarke. δ-complete decision procedures for satisfiability over the reals. In *International Joint Conference on Automated Reasoning*, pages 286–300. Springer, 2012.

[50] Sicun Gao, Jeremy Avigad, and Edmund M Clarke. Delta-decidability over the reals. In *Logic in Computer Science (LICS), 2012 27th Annual IEEE Symposium on*, pages 305–314. IEEE, 2012.

[51] Sicun Gao, Soonho Kong, and Edmund M Clarke. dreal: An smt solver for nonlinear theories over the reals. In *International Conference on Automated Deduction*, pages 208–214. Springer, 2013.

[52] Sicun Gao, Soonho Kong, and Edmund M Clarke. Satisfiability modulo odes. In *Formal Methods in Computer-Aided Design (FMCAD), 2013*, pages 105–112. IEEE, 2013.

[53] Thierry Gautier, Paul Le Guernic, and Loic Besnard. Signal: A declarative language for synchronous programming of real-time systems. In *Conference on Functional Programming Languages and Computer Architecture*, pages 257–277. Springer, 1987.

[54] Antoine Girard and George J Pappas. Approximate bisimulation: A bridge between computer science and control theory. *European Journal of Control*, 17(5-6):568–578, 2011.

[55] Georges Gonthier. A computer-checked proof of the four colour theorem, 2005.

[56] Dimitar P Guelev, Shuling Wang, and Naijun Zhan. Compositional hoare-style reasoning about hybrid csp in the duration calculus. In *International Symposium on Dependable Software Engineering: Theories, Tools, and Applications*, pages 110–127. Springer, 2017.

[57] Ernst Moritz Hahn, Arnd Hartmanns, Holger Hermanns, and Joost-Pieter Katoen. A compositional modelling and analysis framework for stochastic hybrid systems. *Formal Methods in System Design*, 43(2):191–232, 2013.

[58] Nicholas Halbwachs, Paul Caspi, Pascal Raymond, and Daniel Pilaud. The synchronous data flow programming language lustre. *Proceedings of the IEEE*, 79(9):1305–1320, 1991.

[59] Nicolas Halbwachs, Yann-Eric Proy, and Pascal Raymond. Verification of linear hybrid systems by means of convex approximations. In *International Static Analysis Symposium*, pages 223–237. Springer, 1994.

[60] Zhi Han and Bruce Krogh. Reachability analysis of large-scale affine systems using low-dimensional polytopes. *Hybrid Systems: Computation and Control*, pages 287–301, 2006.

[61] David Harel, Dexter Kozen, and Jerzy Tiuryn. Dynamic logic. In *Handbook of philosophical logic*, pages 99–217. Springer, 2001.

[62] Constance L Heitmeyer, BG Labaw, and RD Jeffords. A benchmark for comparing different approaches for specifying and verifying real-time systems. Technical report, NAVAL RESEARCH LAB WASHINGTON DC, 1993.

[63] Thomas A Henzinger. The theory of hybrid automata. In *Verification of Digital and Hybrid Systems*, pages 265–292. Springer, 2000.

[64] Thomas A Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. Hytech: the next generation. In *Real-Time Systems Symposium, 1995. Proceedings., 16th IEEE*, pages 56–65. IEEE, 1995.

[65] Thomas A Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. Hytech: A model checker for hybrid systems. *International Journal on Software Tools for Technology Transfer*, 1(1-2):110–122, 1997.

[66] Charles Antony Richard Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, 1978.

[67] Peter Höfner. *Algebraic calculi for hybrid systems*. BoD–Books on Demand, 2009.

[68] John E Hopcroft, Rajeev Motwani, and Jeffrey D Ullman. Introduction to automata theory, languages, and computation. *Acm Sigact News*, 32(1):60–65, 2001.

[69] Jianghai Hu, John Lygeros, and Shankar Sastry. Towards a theory of stochastic hybrid systems. In *International Workshop on Hybrid Systems: Computation and Control*, pages 160–173. Springer, 2000.

[70] Abrial Jean-Raymond. Modelling in event-b. system and software engineering, 2008.

[71] Jean-Baptiste Jeannin and André Platzer. dtl 2: Differential temporal dynamic logic with nested temporalities for hybrid systems. In *International Joint Conference on Automated Reasoning*, pages 292–306. Springer, 2014.

[72] Henrik Ejersbo Jensen, Kim G Larsen, and Arne Skou. Modelling and analysis of a collision avoidance protocol using spin and uppaal. *BRICS Report Series*, 3(24), 1996.

[73] He Jifeng. From csp to hybrid systems. In *A classical mind*, pages 171–189. Prentice Hall International (UK) Ltd., 1994.

[74] Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, et al. sel4: Formal verification of an os kernel. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pages 207–220. ACM, 2009.

[75] Leslie Lamport. Hybrid systems in tla+. In *Hybrid Systems*, pages 77–102. Springer, 1993.

[76] Leslie Lamport. The temporal logic of actions. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 16(3):872–923, 1994.

[77] Leslie Lamport. *Specifying systems: the TLA+ language and tools for hardware and software engineers*. Addison-Wesley Longman Publishing Co., Inc., 2002.

[78] Kim G Larsen, Paul Pettersson, and Wang Yi. Uppaal in a nutshell. *International journal on software tools for technology transfer*, 1(1-2):134–152, 1997.

[79] Xavier Leroy et al. The compcert verified compiler. *Documentation and user's manual. INRIA Paris-Rocquencourt*, 2012.

[80] Jiang Liu, Jidong Lv, Zhao Quan, Naijun Zhan, Hengjun Zhao, Chaochen Zhou, and Liang Zou. A calculus for hybrid csp. In *Asian Symposium on Programming Languages and Systems*, pages 1–15. Springer, 2010.

[81] Sarah M. Loos and André Platzer. Differential refinement logic. In *LICS*. ACM, 2016.

[82] Simon Lunel, Benoît Boyer, and Jean-Pierre Talpin. Compositional proofs in differential dynamic logic. In Axel Legay and Klaus Schneider, editors, *ACSD*, 2017.

[83] Nancy Lynch, Roberto Segala, and Frits Vaandrager. Hybrid i/o automata. *Information and computation*, 185(1):105–157, 2003.

[84] José Meseguer and Raman Sharykin. Specification and analysis of distributed object-based stochastic hybrid systems. In *International Workshop on Hybrid Systems: Computation and Control*, pages 460–475. Springer, 2006.

[85] Stefan Mitsch and André Platzer. The keymaera x proof ide-concepts on usability in hybrid systems theorem proving. *arXiv preprint arXiv:1701.08469*, 2017.

[86] Stefan Mitsch, Jan-David Quesel, and André Platzer. Refactoring, refinement, and reasoning. In *International Symposium on Formal Methods*, pages 481–496. Springer, 2014.

[87] Andreas Müller, Stefan Mitsch, and André Platzer. Verified traffic networks: component-based verification of cyber-physical flow systems. In *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, pages 757–764. IEEE, 2015.

[88] Andreas Müller, Stefan Mitsch, Werner Retschitzegger, Wieland Schwinger, and André Platzer. A component-based approach to hybrid systems safety verification. In Erika Abraham and Marieke Huisman, editors, *IFM*, volume 9681 of *LNCS*, pages 441–456. Springer, 2016.

[89] Andreas Müller, Stefan Mitsch, Werner Retschitzegger, Wieland Schwinger, and André Platzer. Change and delay contracts for hybrid system component verification. In *International Conference on Fundamental Approaches to Software Engineering*, pages 134–151. Springer, 2017.

[90] Tarik Nahhal and Thao Dang. Test coverage for continuous and hybrid systems. In *International Conference on Computer Aided Verification*, pages 449–462. Springer, 2007.

[91] Corina S Păsăreanu and Willem Visser. Verification of java programs using symbolic execution and invariant generation. In *International SPIN Workshop on Model Checking of Software*, pages 164–181. Springer, 2004.

[92] Lawrence C Paulson. *Isabelle: A generic theorem prover*, volume 828. Springer Science & Business Media, 1994.

[93] Yu Peng, Shuling Wang, Naijun Zhan, and Lijun Zhang. Extending hybrid csp with probability and stochasticity. In *International Symposium on Dependable Software Engineering: Theories, Tools, and Applications*, pages 87–102. Springer, 2015.

[94] André Platzer. Cheat sheet of rules in keymaera. `http://symbolaris.com/info/KeYmaera-cheat.pdf`.

[95] André Platzer. Differential dynamic logic for verifying parametric hybrid systems. In *International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*, pages 216–232. Springer, 2007.

[96] André Platzer. A temporal dynamic logic for verifying hybrid system invariants. In *International Symposium on Logical Foundations of Computer Science*, pages 457–471. Springer, 2007.

[97] André Platzer. Differential dynamic logic for hybrid systems. *J. Autom. Reas.*, 41(2):143–189, 2008.

[98] André Platzer. *Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics*. Springer, Heidelberg, 2010.

[99] André Platzer. Quantified differential dynamic logic for distributed hybrid systems. In *International Workshop on Computer Science Logic*, pages 469–483. Springer, 2010.

[100] André Platzer. Stochastic differential dynamic logic for stochastic hybrid programs. In *International Conference on Automated Deduction*, pages 446–460. Springer, 2011.

[101] André Platzer. The complete proof theory of hybrid systems. In *LICS*, pages 541–550. IEEE, 2012.

[102] André Platzer. Logics of dynamical systems. In *LICS*, pages 13–24. IEEE, 2012.

[103] André Platzer. A uniform substitution calculus for differential dynamic logic. In *International Conference on Automated Deduction*, pages 467–481. Springer, 2015.

[104] André Platzer and Edmund M Clarke. Computing differential invariants of hybrid systems as fixedpoints. In *International Conference on Computer Aided Verification*, pages 176–189. Springer, 2008.

[105] André Platzer and Jan-David Quesel. Keymaera: A hybrid theorem prover for hybrid systems (system description). In *International Joint Conference on Automated Reasoning*, pages 171–178. Springer, 2008.

[106] André Platzer and Jan-David Quesel. European Train Control System: A case study in formal verification. In Karin Breitman and Ana Cavalcanti, editors, *ICFEM*, volume 5885 of *LNCS*, pages 246–265. Springer, 2009.

[107] Amir Pnueli. The temporal logic of programs. In *Foundations of Computer Science, 1977., 18th Annual Symposium on*, pages 46–57. IEEE, 1977.

[108] Vaughan R Pratt. Dynamic logic. *Studies in Logic and the Foundations of Mathematics*, 104:251–261, 1982.

[109] Jan-David Quesel and André Platzer. Playing hybrid games with keymaera. In *International Joint Conference on Automated Reasoning*, pages 439–453. Springer, 2012.

[110] Anders P. Ravn, Hans Rischel, and Kirsten Mark Hansen. Specifying and verifying requirements of real-time systems. *IEEE Transactions on Software Engineering*, 19(1):41–55, 1993.

[111] Daniel Ricketts, Gregory Malecha, Mario M Alvarez, Vignesh Gowda, and Sorin Lerner. Towards verification of hybrid systems in a foundational proof assistant. In *Formal Methods and Models for Codesign (MEMOCODE), 2015 ACM/IEEE International Conference on*, pages 248–257. IEEE, 2015.

[112] Daniel Ricketts, Gregory Malecha, and Sorin Lerner. Modular deductive verification of sampled-data systems. In *Proceedings of the 13th International Conference on Embedded Software*, page 17. ACM, 2016.

[113] Mauno Rönkkö and Xuandong Li. Linear hybrid action systems. *Nordic Journal of Computing*, 8(1):159–177, 2001.

[114] Mauno Rönkkö and Anders P Ravn. Action systems with continuous behaviour. In *International Hybrid Systems Workshop*, pages 304–323. Springer, 1997.

[115] Mauno Rönkkö, Anders P Ravn, and Kaisa Sere. Hybrid action systems. *Theoretical Computer Science*, 290(1):937–973, 2003.

[116] Mauno Rönkkö and Kaisa Sere. Refinement and continuous behaviour. In *International Workshop on Hybrid Systems: Computation and Control*, pages 223–237. Springer, 1999.

[117] Ivan Ruchkin, Dionisio De Niz, Sagar Chaki, and David Garlan. Contract-based integration of cyber-physical analyses. In *Embedded Software (EMSOFT), 2014 International Conference on*, pages 1–10. IEEE, 2014.

[118] Ivan Ruchkin, Bradley Schmerl, and David Garlan. Architectural abstractions for hybrid programs. In *Proceedings of the 18th International ACM SIGSOFT Symposium on Component-Based Software Engineering*, pages 65–74. ACM, 2015.

[119] Alberto Sangiovanni-Vincentelli, Werner Damm, and Roberto Passerone. Taming dr. frankenstein: Contract-based design for cyber-physical systems. *European journal of control*, 18(3):217–238, 2012.

[120] Sriram Sankaranarayanan, Henny B Sipma, and Zohar Manna. Constructing invariants for hybrid systems. In *International Workshop on Hybrid Systems: Computation and Control*, pages 539–554. Springer, 2004.

[121] Jeremy Sproston. Decidable model checking of probabilistic hybrid automata. In *International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*, pages 31–45. Springer, 2000.

[122] Alfred Tarski. A decision method for elementary algebra and geometry. In *Quantifier elimination and cylindrical algebraic decomposition*, pages 24–84. Springer, 1951.

[123] Shuling Wang, Naijun Zhan, and Dimitar Guelev. An assume/guarantee based compositional calculus for hybrid csp. In *International Conference on Theory and Applications of Models of Computation*, pages 72–83. Springer, 2012.

[124] Shuling Wang, Naijun Zhan, and Lijun Zhang. A compositional modelling and verification framework for stochastic hybrid systems. *Formal Aspects of Computing*, 29(4):751–775, 2017.

[125] Shuling Wang, Naijun Zhan, and Liang Zou. An improved hhl prover: an interactive theorem prover for hybrid systems. In *International Conference on Formal Engineering Methods*, pages 382–399. Springer, 2015.

[126] Stephen Wolfram. *The mathematica*. Cambridge university press Cambridge, 1999.

[127] Gaogao Yan, Li Jiao, Yangjia Li, Shuling Wang, and Naijun Zhan. Approximate bisimulation and discretization of hybrid csp. In *International Symposium on Formal Methods*, pages 702–720. Springer, 2016.

[128] Gaogao Yan, Li Jiao, Shuling Wang, and Naijun Zhan. Synthesizing systemc code from delay hybrid csp. In *Asian Symposium on Programming Languages and Systems*, pages 21–41. Springer, 2017.

[129] Lijun Zhang, Zhikun She, Stefan Ratschan, Holger Hermanns, and Ernst Moritz Hahn. Safety verification for probabilistic hybrid systems. In *International Conference on Computer Aided Verification*, pages 196–211. Springer, 2010.

[130] Hengjun Zhao, Mengfei Yang, Naijun Zhan, Bin Gu, Liang Zou, and Yao Chen. Formal verification of a descent guidance control program of a lunar lander. In *International Symposium on Formal Methods*, pages 733–748. Springer, 2014.

[131] Liang Zou, Naijun Zhan, Shuling Wang, and Martin Fränzle. Formal verification of simulink/stateflow diagrams. In *International Symposium on Automated Technology for Verification and Analysis*, pages 464–481. Springer, 2015.

[132] Liang Zou, Naijun Zhan, Shuling Wang, Martin Fränzle, and Shengchao Qin. Verifying simulink diagrams via a hybrid hoare logic prover. In *Proceedings of the Eleventh ACM International Conference on Embedded Software*, page 9. IEEE Press, 2013.

[133] Paolo Zuliani, André Platzer, and Edmund M Clarke. Bayesian statistical model checking with application to simulink/stateflow verification. In *Proceedings of the 13th ACM international conference on Hybrid systems: computation and control*, pages 243–252. ACM, 2010.

Models in KeYmaera X

Fibonnaci

```
/* .kyx file for the Fibonnaci function */

ProgramVariables.
 R Fn.
 R Fn1.
 R Fn2.
End.


Problem.
 Fn = 0 & Fn1 = 1 & Fn2 = 1 /* Initial value */
 -> [{Fn := Fn1; Fn1 := Fn2; Fn2 := Fn1 + Fn;}*] /* Program*/
 Fn2 = Fn1 + Fn /* Invariant */
End.

/* Proved by tactic :
implyR(1) ; loop({`Fn2=Fn1+Fn`}, 1) ; <(
  QE,
  closeId,
  composeb(1) ; assignb(1) ; composeb(1) ; assignb(1) ; assignb(1) ; QE
  )
*/
```

Cruise-control

Cruise-control: Engine

```
 ProgramVariables.
 R t. /* time */
 R ttach. /* timestamp of the last execution of the tachymeter */
 R a. /* acceleration set by the tachymeter */
 R stach. /* speed choosen by the tachymeter */
 R sacm. /* speed measured by the tachymeter */
 R seng. /* speed of the engine */
 R eps. /* maximum time between two executions of the tachymeter */
 R S. /* maximum speed */
End.

Problem.
 eps > 0 & S > 0 /* parameters value */
 & t = ttach & sacm = seng /* initial values of t and seng */
 & a = (stach - sacm) / eps /* assumption from the tachymeter */
 & sacm >= 0 & S >= sacm    /* assumption from the tachymeter */
 & stach >= 0 & S >= stach  /* assumption from the tachymeter */
  -> [{seng' = a, t' = 1 /* behavior of the engine
       & (eps >= t - ttach & t - ttach >= 0)} /* evolution domain */
         ] (seng >= 0 & S >= seng) /* guarantees of the engine */
End.

/* Proved by the following tactic
prop ; dC({`seng=a*(t-ttach)+sacm`}, 1) ; <(
  master,
  master
  )
  */
```

Cruise-control: Tachymeter

```
 ProgramVariables.
 R t. /* time */
 R ttach. /* timestamp of the last execution of the tachymeter */
 R a. /* acceleration set by the tachymeter */
 R stach. /* speed choosen by the tachymeter */
 R sacm. /* speed measured by the tachymeter */
 R seng. /* speed of the engine */
 R eps. /* maximum time between two executions of the tachymeter */
 R S. /* maximum speed */
 R delta.
End.

Problem.
 eps > 0 & S > 0 & delta > 0 /* parameters value */
 & stach = 0 & sacm = 0 & a = 0 /* initial values of stach, sacm and a */
 & seng >= 0 & S >= seng /* assumption from the engine */
  -> [ {
       /* behavior of the tachymeter */
        ? (ttach + eps >= t); sacm := seng; stach := *;
        ? (stach >= 0 & S >= stach
         & stach - sacm >= - delta & delta >= stach - sacm);
        a := ((stach - sacm) / eps); ttach := t;
         }*
        ]
        /* guarantees of the tachymeter */
        (sacm >= 0 & S >= sacm
         & stach >= 0 & S >= stach
         & a = ((stach - sacm) / eps))
End.

/* Proved by the followign tactic :
prop ; loop({`sacm>=0&S>=sacm&stach>=0&S>=stach&a=(stach-sacm)/eps`}, 1) ; <(
  master,
  master,
  prop ; master
  )
*/
```

Water-plant

First water-level

```
ProgramVariables.
 R wl1. /* water level of the first tank */
 R wlm1. /* water level measured by the first controller */
 R fin. /* inlet flow of the first tank */
 R t. /* time */
 R tctrl1. /* timestamp of the last execution of the first controller */
 R delta1. /* execution period of the first controller */
 R fout1. /* outlet flow of the first tank */
End.


Problem.
   (fout1 = 0.75 & delta1 = 0.2) /* Environment */
 /* Initial values */
  & tctrl1 = t /* initial value of t and tctrl1 */
 /* Assumptions from the first water-level controller */
    /* value of fin according to the measured water-level */
  & (((wlm1 >= 6.5 -> fin = 0) & (3.5 >= wlm1 -> fin = 1)
  & ((wlm1 > 3.5 & 6.5 > wlm1) -> (fin = 0 | fin = 1)))
    /* assumption that the measured water-level is in the range [3,7] */
  & (wlm1 >= 3 & 7 >= wlm1)
    /* solution of the differential equation */
  & wl1=(fin-fout1)*(t-tctrl1)+wlm1)
   ->
         [
 /* Behavior of the water-level */
          {wl1' = fin - fout1, t' = 1 /* differential equation */
          & (t >= 0 & wl1 >= 0 & t - tctrl1 >= 0 & delta1 >= t - tctrl1)
          /* evolution domain */
          }
         ]
 /* Guarantees */
         ((wl1 >= 3 & 7 >= wl1)  /* the water-level stay in the desired range */
         & wl1=(fin-fout1)*(t-tctrl1)+wlm1)
End.

/* Proved by the tactic master */
```

Second water-level

```
ProgramVariables.
 R wl2. /* water level of the second tank */
 R wlm2. /* water level measured by the second controller */
 R fout1. /* inlet flow of the second tank */
 R t. /* time */
 R tctrl2. /* timestamp of the last execution of the second controller */
 R delta2. /* execution period of the second controller */
 R fout2. /* outlet flow of the second tank */
End.

Problem.
   (fout1 = 0.75 & delta2 = 0.2) /* Environment */
  /* Initial values */
   & tctrl2 = t /* initial value of t and tctrl2 */
  /* Assumptions from the second water-level controller */
     /* value of fout2 according to the measured water-level */
   & (((wlm2 >= 6.5 -> fout2 = 1) & (3.5 >= wlm2 -> fout2 = 0)
   & ((wlm2 > 3.5 & 6.5 > wlm2) -> (fout2 = 0 | fout2 = 1)))
     /* assumption that the measured water-level is within the range [3,7] */
   & (wlm2 >= 3 & 7 >= wlm2)
     /* solution of the differential equation */
   & wl2=(fout1-fout2)*(t-tctrl2)+wlm2)
    ->
          [
  /* Behavior of the water-level */
           {wl2' = fout1 - fout2, t' = 1 /* differential equation */
           & (t >= 0 & wl2 >= 0 & t - tctrl2 >= 0 & delta2 >= t - tctrl2) /* evolution doma
           }
          ]
  /* Guarantees */
          ((wl2 >= 3 & 7 >= wl2) /* the water-level stay in the desired range */
          & wl2=(fout1-fout2)*(t-tctrl2)+wlm2)
End.


/* Proved by tactic master*/
```

First water-level controller

```
ProgramVariables.
 R wl1. /* water level of the first tank */
 R wlm1. /* water level measured by the first controller */
 R fin. /* inlet flow of the first tank */
 R t. /* time */
 R tctrl1. /* timestamp of the last execution of the first controller */
 R delta1. /* execution period of the first controller */
 R fout1. /* outlet flow of the first tank */
End.

Problem.
 /* Environment */
     delta1 = 0.2
 /* Initial value */
   & wlm1 >= 3 & 7 >= wlm1 /* initial assumption on the measured water-level */
   /* initial assumption on the relation between the measured water-level and inlet flow */
   & ((wlm1 >= 6.5 -> fin = 0) & (3.5 >= wlm1 -> fin = 1)
   & ((wlm1 > 3.5 & 6.5 > wlm1) -> (fin = 0 | fin = 1)))
 /* Assumption from the first water-level */
     /* the first water-level is in the range [3,7] */
   & wl1 >=3 & 7 >= wl1
     /* solution of the differential equation */
   & wl1=(fin-fout1)*(t-tctrl1)+wlm1
   -> [
 /* Behavior of the controller */
       {
        ?tctrl1 + delta1 >= t;wlm1:=wl1;
        {?wlm1>=6.5;fin:=0; ++ ?3.5>=wlm1;fin:=1;}
        ;tctrl1:=t;
       }*
      ]
 /* Guarantees of the controller */
       /* behavior of the controller according to the value of the measured water-level */
       ((((wlm1 >= 6.5 -> fin = 0) & (3.5 >= wlm1 -> fin = 1)
       & ((wlm1 > 3.5 & 6.5 > wlm1) -> (fin = 0 | fin = 1)))
       /* measured water-level is in the range [3,7] */
       & (wlm1 >= 3 & 7 >= wlm1))
       /* solution of the differential equation */
       & wl1=(fin-fout1)*(t-tctrl1)+wlm1)
End.

/* Proved by tactic master */
```

Second water-level controller

```
ProgramVariables.
 R wl2. /* water level of the second tank */
 R wlm2. /* water level measured by the second controller */
 R fout1. /* inlet flow of the second tank */
 R t. /* time */
 R tctrl2. /* timestamp of the last execution of the second controller */
 R delta2. /* execution period of the second controller */
 R fout2. /* outlet flow of the second tank */
End.

Problem.
 /* Environment */
     delta2 = 0.2
 /* Initial values */
    & wlm2 >= 3 & 7 >= wlm2 /* initial assumption on the measured water-level */
    /* initial assumption on the relation between the measured water-level and inlet flow
    & ((wlm2 >= 6.5 -> fout2 = 1) & (3.5 >= wlm2 -> fout2 = 0)
    & ((wlm2 > 3.5 & 6.5 > wlm2) -> (fout2 = 0 | fout2 = 1)))
 /* Assumption from the second water-level */
     /* the second water-level is in the range [3,7] */
    & wl2 >=3 & 7 >= wl2
     /* solution of the differential equation */
    & wl2=(fout1-fout2)*(t-tctrl2)+wlm2
   -> [
 /* Behavior of the controller */
       {
          ?tctrl2 + delta2 >= t;wlm2:=wl2;
          {?wlm2>=6.5;fout2:=1; ++ ?3.5>=wlm2;fout2:=0;}
          ;tctrl2:=t;
        }*
       ]
 /* Guarantees of the controller */
        /* behavior of the controller according to the value of the measured water-level *
          ((((wlm2 >= 6.5 -> fout2 = 1) & (3.5 >= wlm2 -> fout2 = 0)
          & ((wlm2 > 3.5 & 6.5 > wlm2) -> (fout2 = 0 | fout2 = 1)))
        /* measured water-level is in the range [3,7] */
        & (wlm2 >= 3 & 7 >= wlm2))
        /* solution of the differential equation */
        & wl2=(fout1-fout2)*(t-tctrl2)+wlm2)
End.

/* Proved by tactic master */
```

First water-tank

```
ProgramVariables.
 R wl1. /* water level of the first tank */
 R wlm1. /* water level measured by the first controller */
 R fin. /* inlet flow of the first tank */
 R t. /* time */
 R tctrl1. /* timestamp of the last execution of the first controller */
 R delta1. /* execution period of the first controller */
 R fout1. /* outlet flow of the first tank */
End.

Problem.
  /* Environment */
   (delta1 = 0.2 & fout1 = 0.75)
  /* Assumptions of the plant */
    /* value of fin according to the measured water-level */
  & (((wlm1 >= 6.5 -> fin = 0) & (3.5 >= wlm1 -> fin = 1)
  & ((wlm1 > 3.5 & 6.5 > wlm1) -> (fin = 0 | fin = 1)))
    /* assumption that the measured water-level is in the range [3,7] */
  & (wlm1 >= 3 & 7 >= wlm1)
  /* Assumption of the controller */
     /* the first water-level is in the range [3,7] */
  & wl1 >=3 & 7 >= wl1
  /* solution of the differential equation */
   & wl1=(fin-fout1)*(t-tctrl1)+wlm1
   -> [
   /* Behavior obtained by parallel composition */
       {
        /* behavior of the first water-level controller */
          {?tctrl1 + delta1 >= t;wlm1:=wl1;
          {?wlm1>=6.5;fin:=0; ++ ?3.5>=wlm1;fin:=1;}
          ;tctrl1:=t;
           }
          /* behavior of the water-level */
     ++ {wl1' = fin - fout1, t' = 1
          & (t >= 0 & wl1 >= 0 & t - tctrl1 >= 0 & delta1 >= t - tctrl1)
        }
       }*
      ]
    /* Guarantees of the first controller */
       /* behavior of the controller according to the value of the measured water-level */
       ((((wlm1 >= 6.5 -> fin = 0) & (3.5 >= wlm1 -> fin = 1)
       & ((wlm1 > 3.5 & 6.5 > wlm1) -> (fin = 0 | fin = 1)))
       /* measured water-level is in the range [3,7] */
       & (wlm1 >= 3 & 7 >= wlm1))
    /* Guarantees of the first water-level */
```

```
      &((wl1 >= 3 & 7 >= wl1)  /* the water-level stay in the desired range */
    /* Solution of the differential equation */
    & wl1=(fin-fout1)*(t-tctrl1)+wlm1)
End.

/* Proved by tactic
 implyR(1) ; loop({`(((wlm1>=6.5->fin=0)&(3.5>=wlm1->fin=1)&(wlm1>3.5&6.5>wlm1->fin=0|fin=
  master,
  master,
  choiceb(1) ; andL(-1) ; andL(-5) ; andR(1) ; <(
    boxAnd(1) ; andR(1) ; <(
      master,
      boxAnd(1) ; andR(1) ; <(
        GV(1) ; master,
        master
        )
      ),
    boxAnd(1) ; andR(1) ; <(
      GV(1) ; master,
      master
      )
    )
  )
  */
```

Second water-tank

```
ProgramVariables.
 R wl2. /* water level of the second tank */
 R wlm2. /* water level measured by the second controller */
 R fout1. /* inlet flow of the second tank */
 R t. /* time */
 R tctrl2. /* timestamp of the last execution of the second controller */
 R delta2. /* execution period of the second controller */
 R fout2. /* outlet flow of the second tank */
End.

Problem.
 /* Environment */
   (delta2 = 0.2 & fout1 = 0.75)
 /* Assumptions of the plant */
  /* value of fout2 according to the second measured water-level */
   & (((wlm2 >= 6.5 -> fout2 = 1) & (3.5 >= wlm2 -> fout2 = 0)
   & ((wlm2 > 3.5 & 6.5 > wlm2) -> (fout2 = 0 | fout2 = 1)))
   /* assumption that the measured water-level is in the range [3,7] */
   & (wlm2 >= 3 & 7 >= wlm2))
 /* Assumptions of the controller */
   /* the second water-level is in the range [3,7] */
   & (wl2 >=3 & 7 >= wl2)
   /* solution of the differential equation */
   & wl2=(fout1-fout2)*(t-tctrl2)+wlm2
   -> [
   /* Behavior obtained by parallel composition */
       {
       /* behavior of the second water-level controller */
          {?tctrl2 + delta2 >= t;wlm2:=wl2;
           {?wlm2>=6.5;fout2:=1; ++ ?3.5>=wlm2;fout2:=0;}
           ;tctrl2:=t;
           }
       /* behavior of the second water-level */
     ++ {wl2' = fout1 - fout2, t' = 1
          & (t >= 0 & wl2 >= 0 & t - tctrl2 >= 0 & delta2 >= t - tctrl2)
          }
          }*
       ]
     /* Guarantees of the controller */
      /* behavior of the controller according to the value of the measured water-level */
      ((((wlm2 >= 6.5 -> fout2 = 1) & (3.5 >= wlm2 -> fout2 = 0)
      & ((wlm2 > 3.5 & 6.5 > wlm2) -> (fout2 = 0 | fout2 = 1)))
      /* measured water-level is in the range [3,7] */
      & (wlm2 >= 3 & 7 >= wlm2))
     /* Guarantees of the water-level */
```

```
      & (wl2 >= 3 & 7 >= wl2)
     /* Solution of the differential equation */
      & wl2=(fout1-fout2)*(t-tctrl2)+wlm2)
End.

/*
Proved by tactic
implyR(1) ; loop({`(((wlm2>=6.5->fout2=1)&(3.5>=wlm2->fout2=0)&(wlm2>3.5&6.5>wlm2->fout2=(
  master,
  master,
  andL(-1) ; andL(-5) ; choiceb(1) ; andR(1) ; <(
    boxAnd(1) ; andR(1) ; <(
      master,
      boxAnd(1) ; andR(1) ; <(
        GV(1) ; master,
        master
        )
      ),
    boxAnd(1) ; andR(1) ; <(
      GV(1) ; master,
      master
      )
    )
  )

 */
```

Water-plant

```
ProgramVariables.
 R wl1. /* water level of the first tank */
 R wlm1. /* water level measured by the first controller */
 R fin. /* inlet flow of the first tank */
 R t. /* time */
 R tctrl1. /* timestamp of the last execution of the first controller */
 R delta1. /* execution period of the first controller */
 R fout1. /* outlet flow of the first tank */
 R wl2. /* water level of the second tank */
 R wlm2. /* water level measured by the second controller */
 R tctrl2. /* timestamp of the last execution of the second controller */
 R delta2. /* execution period of the second controller */
 R fout2. /* outlet flow of the second tank */
End.

Problem.
 /* Environment */
   (delta1 = 0.2 & fout1 = 0.75 & delta2 = 0.2)
 /* Assumptions of the first water-tank */
   /* value of fin according to the measured water-level */
  & (((wlm1 >= 6.5 -> fin = 0) & (3.5 >= wlm1 -> fin = 1)
  & ((wlm1 > 3.5 & 6.5 > wlm1) -> (fin = 0 | fin = 1)))
    /* assumption that the measured water-level is in the range [3,7] */
  & (wlm1 >= 3 & 7 >= wlm1)
  /* Assumption of the controller */
     /* the first water-level is in the range [3,7] */
  & wl1 >=3 & 7 >= wl1
  /* solution of the differential equation */
   & wl1=(fin-fout1)*(t-tctrl1)+wlm1
/* Assumptions of the second water-tank */
   /* Assumptions of the plant */
    /* value of fout2 according to the second measured water-level */
   & (((wlm2 >= 6.5 -> fout2 = 1) & (3.5 >= wlm2 -> fout2 = 0)
   & ((wlm2 > 3.5 & 6.5 > wlm2) -> (fout2 = 0 | fout2 = 1)))
   /* assumption that the measured water-level is in the range [3,7] */
   & (wlm2 >= 3 & 7 >= wlm2))
  /* Assumptions of the controller */
   /* the second water-level is in the range [3,7] */
   & (wl2 >=3 & 7 >= wl2)
   /* solution of the differential equation */
   & wl2=(fout1-fout2)*(t-tctrl2)+wlm2
  -> [
    /* Behavior obtained by parallel composition */
      {
      /* behavior of the first water-level controller */
```

```
        {?tctrl1 + delta1 >= t;wlm1:=wl1;
         {?wlm1>=6.5;fin:=0; ++ ?3.5>=wlm1;fin:=1;}
         ;tctrl1:=t;
           }
       /* behavior of the second water-level controller */
    ++ {?tctrl2 + delta2 >= t;wlm2:=wl2;
         {?wlm2>=6.5;fout2:=1; ++ ?3.5>=wlm2;fout2:=0;}
         ;tctrl2:=t;
           }
       /* parallel composition of two water-levels */
    ++ {wl1' = fin - fout1, wl2' = fout1 - fout2, t' = 1
          & (t >= 0 & wl1 >= 0 & t - tctrl1 >= 0 & delta1 >= t - tctrl1
             & wl2 >= 0 & t - tctrl2 >= 0 & delta2 >= t - tctrl2)
         }
        }*
       ]
    /* Guarantees of the first controller */
      /* behavior of the controller according to the value of the measured water-level */
           (((((wlm1 >= 6.5 -> fin = 0) & (3.5 >= wlm1 -> fin = 1)
           & ((wlm1 > 3.5 & 6.5 > wlm1) -> (fin = 0 | fin = 1)))
      /* the measured water-level is in the range [3,7] */
         & (wlm1 >= 3 & 7 >= wlm1))
    /* Guarantees of the first water-level */
        &((wl1 >= 3 & 7 >= wl1)  /* the water-level stay in the desired range */
     /* Solution of the differential equation */
        & wl1=(fin-fout1)*(t-tctrl1)+wlm1))
     /* Guarantees of the controller */
       /* behavior of the controller according to the value of the measured water-level */
        & ((((wlm2 >= 6.5 -> fout2 = 1) & (3.5 >= wlm2 -> fout2 = 0)
        & ((wlm2 > 3.5 & 6.5 > wlm2) -> (fout2 = 0 | fout2 = 1)))
       /* measured water-level is in the range [3,7] */
        & (wlm2 >= 3 & 7 >= wlm2))
     /* Guarantees of the water-level */
        & (wl2 >= 3 & 7 >= wl2)
      /* Solution of the differential equation */
        & wl2=(fout1-fout2)*(t-tctrl2)+wlm2))
End.

/* Proved by tactic
implyR(1) ; loop({`((((wlm1>=6.5->fin=0)&(3.5>=wlm1->fin=1)&(wlm1>3.5&6.5>wlm1->fin=0|fin=
  master,
  closeId,
  andL(-1) ; andL(-5) ; andL(-4) ; andL(-5) ; andL(-6) ; choiceb(1) ; andR(1) ; <(
    boxAnd(1) ; andR(1) ; <(
      boxAnd(1) ; andR(1) ; <(
        master,
        boxAnd(1) ; andR(1) ; <(
```

```
        GV(1) ; master,
        master
        )
      ),
    GV(1) ; master
    ),
  choiceb(1) ; andR(1) ; <(
    boxAnd(1) ; andR(1) ; <(
      GV(1) ; master,
      boxAnd(1) ; andR(1) ; <(
        master,
        boxAnd(1) ; andR(1) ; <(
          GV(1) ; master,
          master
          )
        )
      ),
    boxAnd(1) ; andR(1) ; <(
      boxAnd(1) ; andR(1) ; <(
        GV(1) ; master,
        master
        ),
      boxAnd(1) ; andR(1) ; <(
        GV(1) ; master,
        master
        )
      )
    )
  )
)
*/
```