University of Zagreb

FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

Slaven Glumac

AUTOMATED CONFIGURING OF NON-ITERATIVE CO-SIMULATION MODELED BY SYNCHRONOUS DATA FLOW

DOCTORAL THESIS

Zagreb, 2022

University of Zagreb

FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

Slaven Glumac

AUTOMATED CONFIGURING OF NON-ITERATIVE CO-SIMULATION MODELED BY SYNCHRONOUS DATA FLOW

DOCTORAL THESIS

Supervisor: Professor Zdenko Kovačić, PhD

Zagreb, 2022

Sveučilište u Zagrebu

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Slaven Glumac

AUTOMATIZIRANO OBLIKOVANJE NEITERATIVNE KOSIMULACIJE MODELIRANE SINKRONIM PROTOKOM PODATAKA

DOKTORSKI RAD

Mentor: Prof. dr. sc. Zdenko Kovačić

Zagreb, 2022

Doctoral thesis has been made at the University of Zagreb, Faculty of Electrical Engineering and Computing, Department of Control and Computer Engineering.

Mentor: Professor Zdenko Kovačić, Ph.D.

Doctoral thesis has: 131 pages

Number of doctoral thesis: ___________

About the Supervisor

Zdenko Kovačić (1958) is a full professor at the University of Zagreb Faculty of Electrical Engineering and Computing. He is a founder and head of the Laboratory for Robotics and Intelligent Control Systems (LARICS) since 1996. He spent 1990/91 as an IREX visiting researcher at the Bradley Department of Electrical Engineering, the Virginia Polytechnic Institute and the State University, Blacksburg, USA. From 2004-2008, he was head of the Department of Control and Computer Engineering. In 2013, he was awarded the "Fran Bošnjaković" prize by the University of Zagreb for his special scientific efforts in the promotion and development of control, flexible automation and robotics systems. He is also the recipient of the Faculty of Electrical Engineering and Computing "Josip Lončar" Award for 2018. In his 37-year academic career, he supervised more than 100 Bachelor of Science, 150 Master of Engineering, 7 Master of Science and 8 Doctor of Science degrees. He currently supervises 20 undergraduate and 5 graduate students. He has been the principal investigator of more than forty successfully completed international and Croatian R&D projects. Currently, he is the leader of three R&D projects with industrial partners, coordinator and team leader of two ESIF-funded R&D projects, and collaborator on several ongoing Horizon 2020 projects, including DATACROSS, AEROTWIN, ENCORE, and AerialCore. He is also a participant and member of the management committee of the COST Action CA19104 Advancing Social inclusion through Technology and Empowerment (a- STEP). He has co-authored 2 scientific monographs (international publishers Springer Verlag and Taylor Francis (CRC Press), the latter also published in Chinese), 1 university textbook, all in the areas of robotics, manufacturing systems, and intelligent control. He has more than 200 scientific publications (book chapters, journal articles, conference papers). He has been actively involved in organizing numerous international conferences, workshops and academic seminars. He is a member of the EuRobotics PhD Award Jury (2016-2022). He is the Senior Member of IEEE, member of IFAC TC for Adaptive Control and Tuning. In 2010-2013 he was the President of Croatian IEEE Robotics and Automation Chapter. He was the President of Croatian Robotics Society 2005-2010 (also founder) and Vice President (2011-2014). In 2012-2015 he was the president-elect of Croatian Robotic Association. He is also a founder and member of the Presidium of KoREMA - Croatian Society for Communication, Computer, Electronics, Measurement and Control. He is a member of several editorial boards of journals. He actively participated in the popularization of science by participating in various events such as the Festival of Science (Technical Museum, Zagreb), Festival of technical culture, Robotics Schools organized by Croatian Robotics Society, demonstration seminars for secondary school teachers and others. He is the holder of one patent. He won the gold medal for innovation at the 13th International Fair of Innovations in Agriculture, Food Industry and Agricultural Machinery AGRO ARCA 2022.

O mentoru

Zdenko Kovačić (1958.) redoviti je profesor na Sveučilištu u Zagrebu Fakultetu elektrotehnike i računarstva. Osnivač je i voditelj Laboratorija za robotiku i inteligentne sustave upravljanja (LARICS) od 1996. godine. Proveo je 1990./91. kao gostujući istraživač IREX-a na Bradley Department of Electrical Engineering, the Virginia Polytechnic Institute and the State University, Blacksburg, SAD. Od 2004.-2008. bio je predstojnik Zavoda za automatiku i računalno inženjerstvo. Godine 2013. dobio je nagradu "Fran Bošnjaković" Sveučilišta u Zagrebu za posebno istaknuti znanstveni rad u promicanju i razvoju sustava upravljanja, fleksibilne automatizacije i robotike. Dobitnik je i nagrade Fakulteta elektrotehnike i računarstva "Josip Lončar" za 2018. U svojoj 37-godišnjoj akademskoj karijeri vodio je više od 100 prvostupnika, 150 magistara inženjera, 7 magistara znanosti i 8 doktora znanosti. Trenutno mentorira 20 studenata preddiplomskih i diplomskih studija te 5 studenata doktorskog studija. Bio je glavni istraživač više od četrdeset uspješno završenih međunarodnih i hrvatskih R&D projekata. Trenutno je voditelj tri razvojno-istraživačka projekta s industrijskim partnerima, koordinator i voditelj tima dva znanstvena projekta financirana od ESIF-a te suradnik na nekoliko tekućih projekata iz programa Obzor 2020, uključujući DATACROSS, AEROTWIN, ENCORE i AerialCore. Također je sudionik i član upravnog odbora COST Action CA19104 Unapređenje socijalne uključenosti kroz tehnologiju i osnaživanje (a-STEP). Koautor je 2 znanstvene monografije međunarodnih izdavača Springer Verlag i Taylor Francis (CRC Press), potonja također objavljena na kineskom jeziku), 1 sveučilišni udžbenik, sve u područjima robotike, proizvodnih sustava i inteligentnog upravljanja. Ima više od 200 znanstvenih publikacija (poglavlja u knjigama, članci u časopisima, radovi sa skupova). Aktivno je sudjelovao u organizaciji brojnih međunarodnih konferencija, radionica i akademskih seminara. Član je žirija za dodjelu doktorske nagrade EuRobotics (2016.-2022.). On je Senior Member član IEEE-a te član IFAC TC-a za adaptivno upravljanje i ugađanje. Od 2010. do 2013. bio je predsjednik hrvatske IEEE sekcije za robotiku i automatizaciju. Bio je predsjednik Hrvatskog robotičkog društva 2005.-2010. (ujedno i osnivač) i dopredsjednik (2011.-2014.). Od 2012. do 2015. bio je izabrani predsjednik Hrvatskog robotičkog saveza. Također je osnivač i član Predsjedništva KoREMA - Hrvatskog društva za komunikacije, računalstvo, elektroniku, mjerenje i upravljanje. Član je nekoliko uredništva časopisa. Aktivno je sudjelovao u popularizaciji znanosti sudjelujući u različitim manifestacijama kao što su Festivali znanosti (Tehnički muzej, Zagreb), Festival tehničke kulture, Škole robotike u organizaciji HDR-a, demonstracijski seminari za srednješkolske profesore i drugo. Nositelj je jednog patenta. Dobitnik je zlatne medalje za inovaciju na 13. međunarodnom sajmu inovacija u poljoprivredi, prehrambenoj industriji i poljoprivrednoj mehanizaciji AGRO ARCA 2022.

Mojim roditeljima Mirjani i Miloradu, i mojoj Nicole za svu ljubav i podršku

Acknowledgements

When I think about how I managed to write a PhD thesis, it's a lot of fortunate circumstances. Some of it is due to my elementary school teachers, Nada Jurki, Dušan Bogovac, some to my chemistry teacher Ana Fusić, people who kept telling me that I could calculate something. Part of this is due to my friend Matija expressing an interest in what I do. Friendships from my high school class helped me, the colleagues I studied in control engineering certainly motivated me. There are many things, some larger, some smaller, that helped me complete this work. I would like to thank everyone who made it possible for me to write this thesis. However, I will try to name the people I am most grateful to.

My biggest thanks go to my parents Mirjana and Milorad Glumac. Without their love and dedication I could not do this work. They worked very hard to ensure that I generally have a good life and not just be able to write a thesis. It is worth noting that I owe them not only love. At the moment of the crisis, my mother told me to finish my thesis. You have to listen to what your mother tells you. I have another example of their direct influence very early in life. My father taught me to multiply very young using his 30 year old pencil case. He knew how to get me to look at the multiplication tables printed on the top of the case. I've heard that this is a requirement for people to learn math. Since there is a lot of math in this work, I think they had a direct impact on the thesis. I want to give them credit for all the theorems in this work because I think they deserve it.

A person who has supported me throughout most of my PhD is my dear Nicole. She deserves an award for patience and support. I met her after I started working on my PhD and we had a joke about how she doesn't know what it's like when I'm not working on my PhD. We're finally going to find out. When I needed her to proofread she would proofread, when I needed to practice my presentation she was a rehearsal audience, when I needed motivation she was there for me. I'm very grateful to her for that, but I'm happy to say that I'm not just grateful for the proofreading. I can say that I love her and I know that she loves me back. When you realize that, everything is easier, even writing a PhD.

I would like to thank my mentor Professor Zdenko Kovačić who supported and mentored me throughout most of my studies. He is the person who most influenced my decision to continue my PhD studies. He gave me the opportunity to work with colleagues with similar interests on interesting topics available in the LARICS laboratory. This sparked my interest in research work and continued when I changed jobs. With all the work he has, he was always there to offer useful advice. He was always understanding and would give me guidance or just review ideas whenever needed. He made my research possible and I want to thank him for that.

I thank AVL for providing the working conditions and support in choosing the topic of the thesis. I would like to thank the Paprika team for being supportive, nice colleagues and just friends in general. They have created a comfortable environment that has produced work where striving for improvement can lead to a PhD. Toni Benussi, Mario Volarević, Tomislav Ljubej, Ivan Kvesić, Fran Raos, Lorena Miletić, Antonio Dropulić, thank you very much! Colleagues from AVL also had a direct influence on the thesis. When I was thinking about a topic, I asked my development owner at the time, Michael Kang, for help. Shortly thereafter, Joseph Zehether pointed out a problem in non-iterative co-simulation that helped shape this thesis. Thank you both very much. At that time, Dietmar Peinsipp was entrusted with supervising my efforts. I would like to thank Dietmar because his supervision just felt like support, be it in research or in a personal sense. It is also worth noting that colleagues were there to provide support when concrete research help was needed. I would like to thank Nikola Varga, Fran Raos and Marin Miletić for helping me to finish the research work for my research papers. Many thanks to the entire AVL.

Finally, I would like to close in a similar way to how I started. I am grateful that I was able to complete my dissertation and that I feel that the environment supports me. I hope that my research will have a positive impact on the lives of those around me.

Abstract

This thesis proposes the use of an SDF graph (SDFG) to model the computation of a non-iterative co-simulation. An SDFG is a determinate model of computation with a lot of scheduling research available. A method for creating an SDFG from a co-simulation network, initial tokens and simulator step sizes is described. A simulator in the proposed method is a synchronous data flow actor that is used to model the execution of a co-simulation slave. It is shown that such an SDFG has consistent rates and uniformly updates the states of all simulators.

To analyze the behavior of numerical errors introduced in the co-simulation, it is assumed that coupled ordinary differential equations can represent the modeled system. This thesis uses the numerical defect analysis to formulate a co-simulation quality criterion. The numerical defect is divided into integration, output and connection defects. Such a defect distribution reflects the division of responsibility between the co-simulation master and the internal solvers of the slaves. This work proposes how connection defects can be calculated and output defects can be estimated. Integration defects are not analyzed as it is assumed that they are controlled by internal solvers of the slaves. The proposed quality criterion for the co-simulation is based on the aggregation of the output and connection defects.

The simulator step sizes, the number and values of the initial tokens should be determined in order to configure the execution of a co-simulation network. A method for calculating the number of initial tokens based on the simulator step sizes is proposed. It is shown that an SDFG with such number of initial tokens does not deadlock. Furthermore, a method how to check whether it can run in real-time is shown. The quality criterion enables an optimization approach to find the values of the initial tokens. In the proposed approach, the quality criterion is assessed in a single iteration of the SDFG. The Nelder-Mead algorithm is used to solve the optimization problem of finding initial token values. To complete the automatic configuration of the co-simulation, the simulator step sizes must be determined. This is done by reducing the simulator step sizes until the requested tolerance is met. This thesis proves that connection and output defects can be controlled by reducing simulator step sizes.

The automated configuration algorithm presented in the last chapter is the main goal of this thesis. The methods for determining the step sizes of the simulator, the number and the values of the initial tokens are building blocks of this algorithm. The validity and usefulness of the proposed algorithm is justified with theorems and examples presented throughout this thesis.

Keywords: co-simulation, synchronous data flow, error analysis, coupled ordinary differential equations, defect control, automated configuration, model-based development

Prošireni sažetak

Automatizirano oblikovanje neiterativne kosimulacije modelirane sinkronim protokom podataka

Uvod

Kosimulacija omogućuje simulaciju složenih sustava povezivanjem različitih simulatora na razini signala. Ova tehnika simulacije je vrlo popularna u praksi jer ne postavlja velike zahtjeve za implementaciju i omogućuje ponovnu upotrebu već razvijenih modela pojedinih komponenti sustava. Radi lakšeg povezivanja komercijalnih simulatora, standardizacija sučelja za simulaciju uvedena je s FMI (engl. Functional Mock-up Interface) standardom. Dodatna prednost kosimulacije je u tome što omogućuje razvoj složenog sustava temeljen na modelu. Faze razvoja temeljenog na modelu su simulacije sa X u petlji (engl. X-in-the-loop simulation), gdje je X komponenta složenog sustava u razvoju. Ova komponenta može biti model, program (engl. software), procesor ili fizička komponenta (engl. hardware), ovisno o trenutnoj fazi razvoja. Ovakav način razvoja omogućuje podjelu rada između razvojnih timova i olakšava njihovu međusobnu komunikaciju. Upotreba simulacije s modelom u petlji brzo otkriva može li proizvod biti razvijen, a zatim daje signalnu specifikaciju za kasnije faze razvoja. U ovom istraživanju razvoj temeljen na modelu započinje sa simulacijom koja u petlji ima model komponente, a završava sa simulacijom koja u petlji ima fizičku komponentu. Ostatak petlje smatra se konstantnim tijekom cijelog razvoja. Specifikacijom složenog sustava u razvoju smatra se mreža podređenih kosimulacijskih jedinica koje implementiraju sučelje prema FMI standardu.

Algoritam upravljanja kosimulacijom (engl. co-simulation master) zadužen je za raspored izvođenja podređenih kosimulacijskih jedinica i aproksimaciju njihovih ulaznih signala. Jedan od ciljeva istraživanja je specificirati algoritam upravljanja kosimulacijom koji može upravljati različitim simulacijama sa X u petlji. Poželjno je da taj algoritam jednoznačno određuje izvođenje podređenih kosimulacijskih jedinica, odnosno da konačna kosimulacija uvijek daje iste rezultate ako slijedi njegovu specifikaciju. Uz jednoznačno određeno izvođenje kosimulacije i konstantan ostatak petlje, različit odziv dvaju faza razvoja temeljenog na modelu nastaje isključivo zbog razlika u obliku komponente. Ovo svojstvo omogućuje korisniku kosimulacije da usporedi razvijenu komponentu s njenim modelom. U praksi se vrlo često mogu pojaviti razlike zbog različitih platformi koje se koriste u različitim fazama razvoja temeljenog na modelu. Prethodni zahtjev može se interpretirati na način da razlike zbog izvođenja na različitim platformama ne bi trebale postojati.

Primjer jednoznačno određenog iterativnog algoritma upravljanja kosimulacijom dostupan je u literaturi. Ovaj algoritam zahtjeva sposobnost podređenih jedinica da ponove korak simulacije i dostupnost informacija o direktnoj izlazno-ulaznoj međuovisnosti. Iako definirane FMI standardom, ove opcije često nisu dostupne u praksi. Nadalje, fizička komponenta ne može ponoviti korak simulacije. Jedan od zahtjeva za specifikaciju praktično primjenjivog algoritma upravljanja kosimulacijom je da od podređenih jedinica očekuje samo obavezne mogućnosti FMI standarda. Nadalje, poželjno je da se njegova primjena može generalizirati na simulaciju s fizičkom komponentom u petlji. Ovi zahtjevi eliminiraju iterativne algoritme upravljanja kosimulacijom i ostavljaju mjesta samo za neiterativne algoritme tipa Gauss-Seidel ili Gauss-Jacobi s jednim ili više taktova simulacije.

Sekvencijalni algoritmi tipa Gauss-Seidel pokazuju nešto bolje rezultate u odnosu na paralelne algoritme tipa Gauss-Jacobi na testnim modelima, ali ograničavaju mogućnosti distribucije izračuna. U slučaju pojedinih platformi, preporučljivo je kombinirati navedene algoritme sa simulacijom u više taktova. Jedan od ciljeva istraživanja je utvrditi može li se specificirati generalizacija navedenih neiterativnih algoritama upravljanja kosimulacijom. Model sinkronog protoka podataka razmotrit će se kao jedan od kandidata za generalizaciju. Model sinkronog protoka podataka jednoznačno je određen model izračuna koji je najčešće korišten za opis fizičkih komponenti. Velika prednost ovog modela je u tome što postoje razvijeni postupci za njegovo raspoređivanje na jedan ili više procesora. Ovaj pristup modeliranju algoritma za upravljanje kosimulacijom spaja dva područja i ima potencijal omogućiti velik broj automatiziranih postupaka koji olakšavaju razvoj temeljen na modelu.

Idealizirani zahtjev za algoritam kosimulacije je minimiziranje globalne pogreške u kosimulaciji. Algoritmi za rješavanje običnih diferencijalnih jednadžbi definiraju približnu mjeru koja dovodi do smanjenja globalne pogreške. Najčešće mjere su lokalna pogreška ili defekt rješavanja diferencijalne jednadžbe. U slučaju kosimulacije, postojeća literatura analizira lokalnu pogrešku, ali ne i numerički defekt. Jedan od ciljeva istraživanja je provjeriti praktičnost upotrebe defekta kao mjere za kvalitetu algoritma upravljanja kosimulacijom. Defekt je korišten kao mjera za kontrolu pogreške za rješavanje skupa diferencijalnih i algebarskih jednadžbi, ali njegova praktičnost za kosimulaciju nije detaljno istražena.

Mjera za procjenu kvalitete algoritma za upravljanje kosimulacijom potrebna je za konačni cilj istraživanja, poboljšanje automatskog traženja kosimulacijske konfiguracije. U postojećim radovima predloženi su postupci za oblikovanje simulacije s modelom u petlji. Ovo istraživanje razmatra prijenos konfiguracije na ostale faze razvoja temeljenog na modelu.

Pregled disertacije

Disertacija je podijeljena u šest poglavlja. Prvo poglavlje daje uvod i motivaciju za automatsko oblikovanje kosimulacije, a posljednje zaključak disertacije. Poglavlja između obrađuju pojedinačne aspekte kosimulacije koji utječu na konačni algoritam automatskog oblikovanja. U disertaciji su navedeni primjeri napisani u programskom jeziku Python.

Prvo poglavlje daje kratak uvod u disertaciju. U navedenom ukratko su opisani problemi u oblikovanju kosimulacije. Nadalje, u poglavlju je opisana motivacija za istraživanje algoritma za automatsko oblikovanje neiterativne kosimulacije. Kroz poglavlje je dan pregled literature u području istraživanja. Poglavlje je zaključeno kratkim pregledom disertacije.

U drugom poglavlju dan je opis dva modela kompleksnog sustava. Prvo je opisan sustav povezanih običnih diferencijalnih jednadžbi. Na temelju ovog modela provedena je analiza utjecaja numeričkog defekta na numeričku pogrešku. Dokazan je teorem koji pokazuje da je numerička pogreška takvog modela ograničena ako je numerički defekt ograničen. Nakon toga je definirana kosimulacijska mreža prema FMI standardu. Opisano je kako se pojmovi u FMI standardu odnose na definiciju u disertaciji. Na kraju drugog poglavlja prikazano je kako se kosimulacijska mreža može koristiti za modeliranje sustava povezanih običnih diferencijalnih jednadžbi. Numerički defekt je podijeljen na integracijski, izlazni i defekt konekcije. Podjela numeričkog defekta omogućuje podjelu odgovornosti između algoritma upravljanja kosimulacijom i algoritama za rješavanje jednadžbi podređenih kosimulacijskih jedinica.

U trećem poglavlju opisan je model sinkronog protoka podataka i njegovu primjenu za modeliranje izvođenja kosimulacije. U prvom dijelu poglavlja model sinkronog protoka podataka strogo je definiran. Nakon toga je naveden primjer grafa sinkronog protoka podataka i definirano njegovo izvođenje. U drugom dijelu poglavlja pokazuje se da je ovaj model jednoznačno određen. Razlika u odnosu na postojeći dokaz u literaturi je složeniji model izvedbe u kojem se operacije dijele na potrošnju, izračun i proizvodnju žetona. U trećem dijelu je predstavljen sekvencijalni algoritam za izvođenje grafa sinkronog protoka podataka. Analiza ovog algoritma daje odgovor na pitanje imaju li međuspremnici ograničen broj žetona tijekom izvršavanja i hoće li doći do zastoja tijekom izvršavanja. Ova je analiza primjenjiva na bilo koji algoritam izvođenja jer je graf sinkronog protoka podataka jednoznačno određen. Treće poglavlje završava opisom kako modelirati izvođenje kosimulacijske mreže pomoću grafa sinkronog protoka podataka. U ovom su dijelu definirani simulatori za izvođenje podređenih kosimulacijskih jedinica i pretvarači stopa potrošnje i proizvodnje žetona. Dokazano je da prilikom izvođenja dobivenog grafa međuspremnici imaju ograničen broj žetona. Također je dokazano da sve podređene kosimulacijske jedinice imaju jednoliku progresiju simuliranog vremena. Ovaj model je bitan doprinos disertacije.

U četvrtom poglavlju opisana je platforma s neograničenim brojem procesora. Na takvoj platformi može se izvršiti paralelan algoritam za izvođenje grafa sinkronog protoka podataka. Takav algoritam opisan je u prvom dijelu poglavlja. Također, opisan je izračun trajanja jedne iteracije grafa sinkronog protoka podataka. Drugi dio opisuje metodu za izračunavanje broja početnih žetona u grafu sinkronog protoka podataka. Ova metoda je cilj četvrtog poglavlja i bitan dio algoritma za automatsko oblikovanje kosimulacije. Dokazano je da graf dobiven na ovaj način ne dovodi do zastoja prilikom izvođenja. Nadalje, dostupna je metoda koja provjerava može li se takav graf izvršiti u stvarnom vremenu. Ova metoda pretpostavlja upotrebu platforme s neograničenim brojem procesora. Ova provjera daje uvid u to može li se izvesti simulacija s fizičkim komponentama u petlji.

U petom poglavlju prikazan je izračun numeričkog defekta. U prvom dijelu prikazan je izračun defekta konekcije. Dokazano je da se defektom konekcije može upravljati smanjenjem komunikacijskog koraka podređenih jedinica. U prvom dijelu prikazana je estimacija izlaznog defekta. Dokazano je da je ova procjena asimptotski ispravna. U drugom poglavlju dokazano je da se izlaznim defektom može upravljati smanjenjem komunikacijskog koraka podređenih jedinica. U trećem dijelu definiran je kriterij za procjenu kvalitete kosimulacije. Kriterij je procijenjen pomoću defekta konekcija i izlaznog defekta. Upravljanje integracijskim defektom prepušteno je podređenim komunikacijskim jedinicama s ovom formulacijom kriterija. Upotreba ovog kriterija opravdana je primjerom u ovom poglavlju i teoremom u drugom poglavlju.

Četvrti dio petog poglavlja opisuje algoritam automatskog oblikovanja za neiterativnu kosimulaciju. Ovaj algoritam je konačni cilj disertacije. Za oblikovanje grafa sinkronog protoka podataka iz kosimulacijske mreže potrebno je odrediti komunikacijske korake simulatora, broj i vrijednosti početnih žetona. Broj početnih žetona određen je metodom prikazanom u četvrtom poglavlju. Vrijednosti početnih žetona određene su pomoću optimizacije na jednoj iteraciji grafa sinkronog protoka podataka. Smanjivanjem komunikacijskih koraka simulatora u algoritmu postiže se željena vrijednost kriterija. Ovaj odjeljak daje primjer upotrebe algoritma automatskog oblikovanja kosimulacije.

Šesto i posljednje poglavlje je zaključak disertacije. U njemu su navedeni doprinosi disertacije i planovi za budući rad.

Zaključak

Glavni cilj disertacije je omogućiti automatsko oblikovanje neiterativne kosimulacije. U sklopu disertacije ostvarena su sljedeća tri izvorna znanstvena doprinosa:

- Model sinkronog protoka podataka za izvo đenje neiterativne kosimulacije.
- Metoda za evaluaciju kosimulacije pomo ću defekta konekcije i izlaznog defekta.
- Metoda automatskog oblikovanja svih faza razvoja temeljenog na modelu pomo ću simulacije s modelom u petlji.

Model sinkronog protoka podataka za izvođenje neiterativne kosimulacije prezentiran je u trećem poglavlju disertacije. Možda najvažnije svojstvo ovog modela je da je njegovo izvođenje jednoznačno određeno. Nadalje, omogućuje upotrebu postojećih algoritama za raspoređivanje zadataka razvijenih za graf sinkronog protoka podataka. U istom poglavlju prikazani su primjeri i teoremi koji pokazuju valjanost predloženog modela. U teoremima je dokazano da su stope proizvodnje i potrošnje u dobivenom grafu sinkronog protoka podataka konzistentne, te da sve podređene kosimulacijske jedinice imaju jednoliku progresiju simuliranog vremena. Ovaj

model je izvorni doprinos i spaja kosimulaciju s istraženim algoritmima za raspoređivanje grafa sinkronog protoka podataka.

Metoda za evaluaciju kosimulacije pomoću defekta konekcije i izlaznog defekta prikazana je u petom poglavlju disertacije. Metoda se sastoji od proračuna defekta konekcije i estimacije izlaznog defekta. Pokazalo se da je estimacija izlaznog defekta asimptotski točna. U drugom je poglavlju dokazano da je kosimulacijska pogreška ograničena ako su integracijski, izlazni i defekt konekcije ograničeni. U disertaciji se pretpostavlja da podređene kosimulacijske jedinice upravljaju integracijskim defektom. Ovo je izvorni doprinos koji pokazuje kako primijeniti numeričku analizu defekta na procjenu kvalitete kosimulacije. U dosadašnjoj literaturi analiza numeričkog defekta provedena je na diferencijalnim i algebarskim jednadžbama, ali ne i na kosimulacijskoj mreži.

Metoda automatskog oblikovanja svih faza razvoja temeljenog na modelu pomoću simulacije s modelom u petlji prikazana je u četvrtom dijelu petog poglavlja. Metoda se sastoji od nekoliko manjih metoda predstavljenih tijekom disertacije. Broj početnih žetona određen je metodom prikazanom u četvrtom poglavlju. Vrijednosti početnih žetona određene su pomoću optimizacije na jednoj iteraciji grafa sinkronog protoka podataka. Smanjivanjem komunikacijskih koraka simulatora u algoritmu postiže se željena vrijednost kriterija. Dokazano je da se smanjenjem komunikacijskih koraka simulatora može postići ograničenje izlaznog i defekta konekcije. Dobiveni graf sinkronog protoka podataka može se primijeniti na druge faze razvoja temeljenog na modelu nakon oblikovanja. U nedavnoj literaturi dostupna je metoda automatskog oblikovanja kosimulacije. Metoda predstavljena u ovoj disertaciji omogućuje postavljanje različitih koraka kosimulacije i omogućuje modeliranje niza složenih sustava.

Ključne riječi: kosimulacija, model sinkronog protoka podataka, analiza pogreške, povezane obične diferencijalne jednadžbe, kontrola defekta, automatizirano oblikovanje, razvoj temeljen na modelu

Contents

Chapter 1

Introduction

Co-simulation enables the simulation of complex systems by connecting different co-simulation slaves at the signal level [1, 2]. This simulation technique is popular in practice because it allows reuse of previously developed subsystem models. With the Functional Mock-up Interface (FMI) standard [3, 4, 5], a standardization of the simulation interface was introduced to facilitate the connection of commercial simulators. The popularity of co-simulation can be assessed by observing the increasing number of tools that support FMI for co-simulation [6]. The main advantage of co-simulation is the distribution of the modeling effort. Each engineering team can develop a co-simulation slave for the respective parts of the system under their responsibility. A subsystem model can be developed in a special simulator and easily connected to other simulators. This is the reason why co-simulation is sometimes referred to as simulator coupling [7]. The terms co-simulation slave and simulator can be used interchangeably. A Functional Mock-

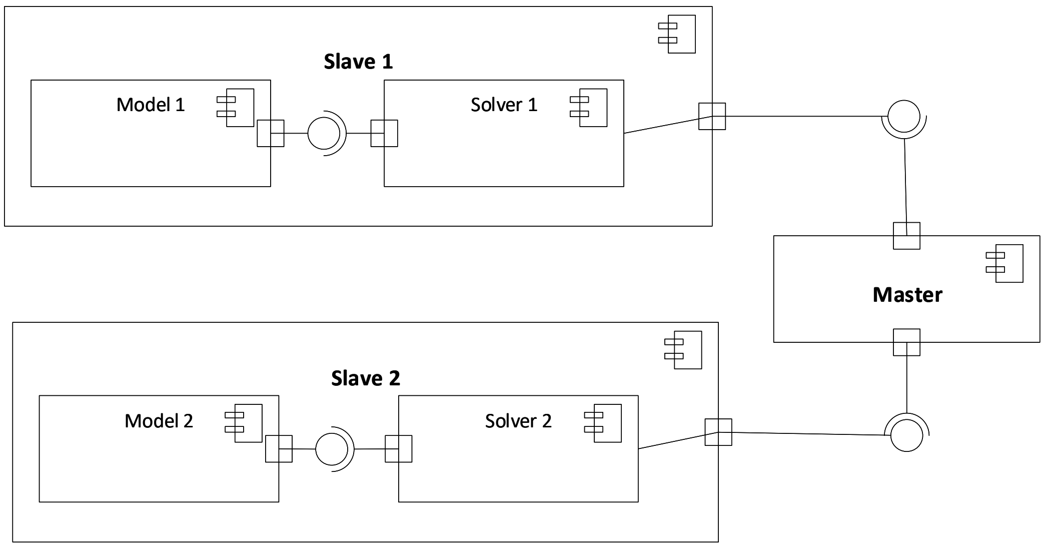


Figure 1.1: A co-simulation slave (CSL) wraps the model of a subsystem and its solver. A co-simulation master (CSM) orchestrates the execution of co-simulation slaves.

up Unit (FMU) for co-simulation is a package that is standardized by the FMI standard [5] for the exchange of co-simulation slaves. The engineering team can ensure that the co-simulation slave has a suitable solver for solving its model. A suitable choice of a co-simulation master (CSM) is then required in order to orchestrate slaves in the co-simulation network (Figure 1.1). The main goal of this work is to automate the selection of a suitable co-simulation master.

A co-simulation network (CSN) corresponds to a co-simulation, just like an ordinary differential equation (ODE) corresponds to a classical simulation. Both are used as simulation models to predict the behavior of a system [8]. However, both can also be used as a specification of a technical system under development. In this case, a CSN becomes a model that is similar to technical drawings for the design of buildings [9]. Such use of co-simulation can be observed in the model-based development (MBD) of complex systems [10, 11, 12, 13, 14]. The specification of a CSN divides the complex system into smaller components that can be developed separately. It can be discussed whether a co-simulation is mainly used as a specification of the system to be developed or as a check to see whether an existing specification is able to meet its requirements[*]. Regardless of the exact perspective of the MBD user, a hardware in the loop (HIL) simulation is usually executed. The results are usually compared to the results of the model in the loop (MIL) simulation. The differences are then analyzed to verify the hardware implementation. In order to use the co-simulation during such a process, the co-simulation should be carried out in real time without repeating simulation steps. This thesis describes a non-iterative co-simulation method that tries to facilitate MBD. A comparison between iterative and non-iterative approaches can be found in [15].

Typically, the first thing that is associated with MBD is a software development process that involves some form of UML modeling [16, 17]. In that context, UML models are used to describe the software system under development from different perspectives [18]. There is a research showing that such kind of MBD is advantageous when the model design costs are low enough compared to the production costs of the system [19, 20]. This thesis tries to improve the development of cyber-physical systems [21, 22, 23] and not pure software systems. Other modeling techniques are being explored within the community researching such systems [24]. These articles highlight the importance of determinacy. Non-determinate programs can have problems such as race conditions [25, 26]. This thesis takes determinacy into account when proposing definitions of models that describe different views of the co-simulation[†]. An attempt is made to precisely define the system model, the model of computation (MOC) and the execution schedule. The precise definition for different layers of the co-simulation architecture enables the exchange of individual layers. The proposed defect analysis method works at the

[*]In practice, the former is usually not explicitly stated, although the modeling of the CSN can be considered as a form of verification.

[†]In the author's opinion, UML is better suited for modeling systems that were developed without a strict software architecture. However, determinate models of computation seem to be more useful for designing new systems.

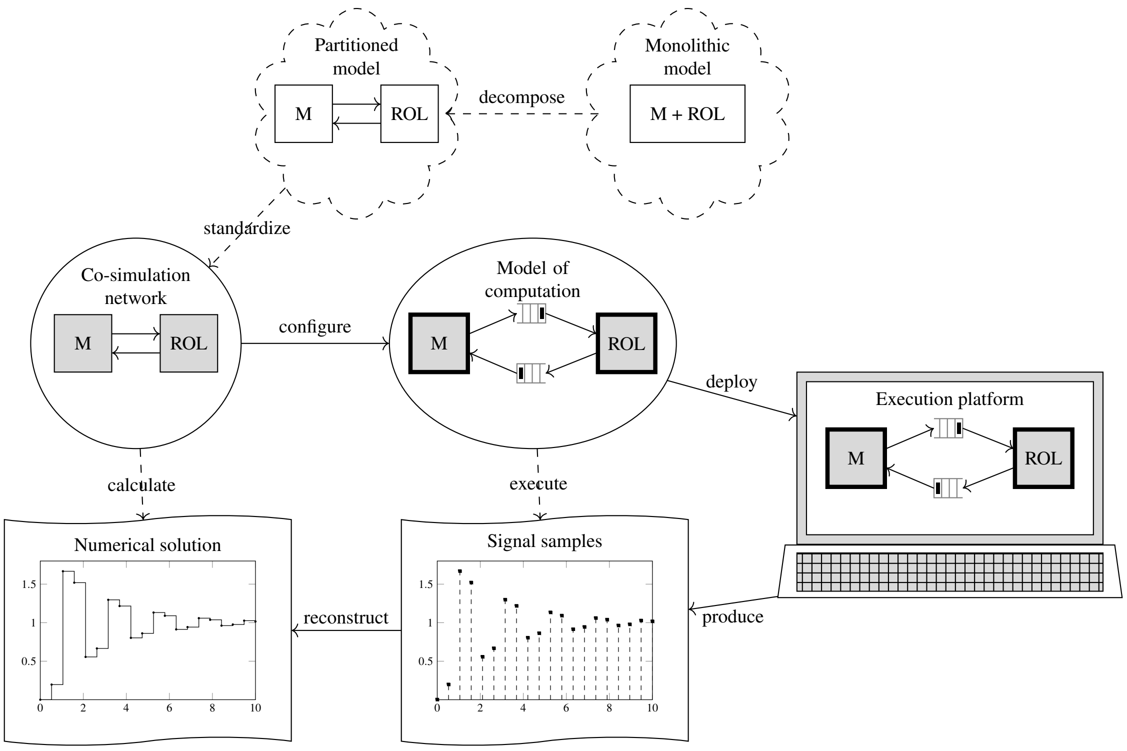


Figure 1.2: The diagram shows the steps for modeling and executing a continuous co-simulation. First steps show the creation of a CSN and the last step leads to a numerical solution of the co-simulation. Various aspects of the co-simulation are presented in intermediate steps

system model level and can be used for both iterative and non-iterative co-simulations. The MOC allows the execution platform to be exchanged. The results should be the same when the co-simulation is run on a personal computer or a computer cluster. It is also interesting to note that SDF can be used to model the behavior of the system developed using MBD [27].

The proposed steps to model and execute a continuous co-simulation are shown informally in Figure 1.2. The diagram shows the steps of the concept phase as cloud shapes. In the concept phase, it should be decided how to decompose the overall system model. After the model decomposition, the subsystem models should be made available via a standardized interface. The interface chosen in this work is FMI for co-simulation. The concept phase added to the diagram represents the underlying assumption subsystem models. In this work it is assumed that the resulting CSN can be represented by the system of coupled ordinary differential equations. This assumption enables a distributed error analysis of a co-simulation.

A piecewise-continuous prediction of the monitored signals is obtained by computing the specified CSN. A CSM must be configured to calculate signals specified by the CSN. This thesis uses SDF [28] as the model of computation (MOC) for the specification of the co-simulation master. The introduction of a MOC enables the execution platform to be exchanged. This means the co-simulation can be executed on a personal computer. Since synchronous data flow (SDF) is a determinate MOC, the results obtained by executing the co-simulation should be exactly the same regardless of the execution algorithm. If the CSN contains a hardware component, the execution of the network can be modeled using SDF. The proposed co-simulation modeled by SDF enables HIL simulation and facilitates MBD.

In the next section, the research motivation is explained and the last section of this chapter describes the outline of the thesis.

1.1 Research motivations

Model-based development (MBD) stages are X in the loop (XIL) simulations, where X is a component of a complex system under development. This component can be a model, software, processor or hardware component, depending on the current stage of development. This mode of development enables distributed development between development teams and facilitates their mutual communication. Using simulation with a model in the loop (MIL) quickly detects whether a product can be developed, and then provides a specification of system behavior for later stages of development. In this research, MBD starts with a model in the loop (MIL) simulation and ends with a hardware in the loop (HIL) simulation. The rest of the loop is considered unchanged during development. The specification of a complex system under development is considered to be a network of CSLs that implement the FMI interface.

The co-simulation master is responsible for scheduling the execution of CSLs and approximating their input signals. One of the goals of the research is to specify a co-simulation master that can handle different XIL simulations. It is desirable that the execution of this algorithm is determinate, i.e. co-simulation always produces the same results if it follows its specification. With determined co-simulation and unchanged rest of the loop, the different response of the two model-based development stages is solely due to differences in exchanged component. This property allows the co-simulation user to compare the developed component with its model. Differences and errors in practice may occur because of the different platforms used at different stages of model-based development.

The previous requirement can be interpreted as saying that performance differences on different platforms should not exist. An example of a determinate co-simulation master is available at [29]. This algorithm requires the ability of slave units to repeat the simulation step and the availability of output-input dependency information. Although defined by the FMI standard, these options are often not available in practice. Furthermore, a hardware component cannot repeat the simulation step. One of the requirements for specifying a practically applicable co-simulation master is that it expects slaves to implement only the mandatory capabilities of the FMI standard. Furthermore, it is desirable that its application can be generalized to HIL simulation. These requirements eliminate iterative co-simulation masters from consideration [30] and leave room only for non-iterative Gauss-Seidel or Jacobi algorithms [31], either single or multi-rate [32, 33].

Sequential Gauss-Seidel algorithms show slightly better results compared to Jacobi parallel algorithms on test models [31], but they limit the distribution possibilities of computation. In the case of single-processor platforms, it is advisable to use the above algorithms with multiple rates of simulation. One of the goals of the research is to determine whether the generalization of the mentioned non-iterative CSMs can be specified. The use of synchronous data flow [28] for this purpose will be investigated. The synchronous data flow (SDF) model is a determinate MOC [34] most commonly used to describe hardware components. A major advantage of this model is that procedures have been developed for executing it in one or more processors [35, 36]. This approach to modeling a co-simulation masters combines two areas of research and has the potential to enable a large number of automated procedures that facilitate model-based development. It is interesting to note that in [37] it has been demonstrated how to wrap an SDFG with an FMU. This thesis goes in the opposite direction to introduce a determinate model of computation (MOC) for non-iterative co-simulation.

Ideally, the co-simulation master should minimize the global simulation error. Algorithms for solving ordinary differential equations always define an approximate measure that leads to the minimization of global error [38]. The most common measures are the local error or numerical defect of solving the differential equation. In the case of co-simulation, the local error was analyzed in [39, 40]. One of the goals of the research is to test the practicality of using the numerical defect as a measure of the quality of the co-simulation master. The defect was used as an error control measure to solve a set of differential and algebraic equations [41], but its convenience for co-simulation has not been explored in detail.

The title of this thesis suggests that an automated configuration algorithm for non-iterative co-simulation is the main goal. Research in this area is limited [42, 43]. The authors in [42] gave an overview of the CSM algorithm, which configures its execution itself in each step. Such an algorithm may not require much configuration by the user. This work shows an algorithm that is similar to the one presented in [43]. In this work, however, a more general underlying model of the system is assumed and the correctness of the algorithm is formally proven.

1.2 Outline of the thesis

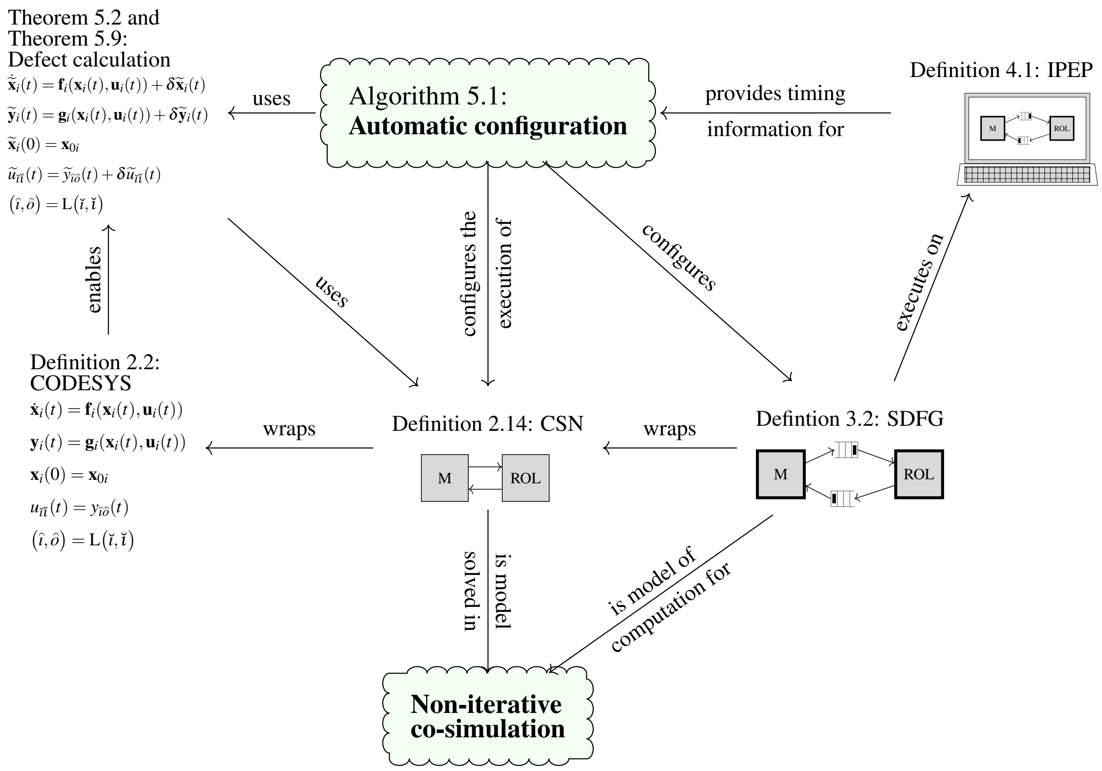

Theorem 5.2 and Theorem 5.9: Defect calculation
$\dot{\widetilde{\mathbf{x}}}_i(t) = \mathbf{f}_i(\mathbf{x}_i(t), \mathbf{u}_i(t)) + \delta\widetilde{\mathbf{x}}_i(t)$
$\widetilde{\mathbf{y}}_i(t) = \mathbf{g}_i(\mathbf{x}_i(t), \mathbf{u}_i(t)) + \delta\widetilde{\mathbf{y}}_i(t)$
$\widetilde{\mathbf{x}}_i(0) = \mathbf{x}_{0i}$
$\widetilde{u}_{\breve{\imath}\check{\imath}}(t) = \widetilde{y}_{\hat{\imath}\hat{o}}(t) + \delta\widetilde{u}_{\breve{\imath}\check{\imath}}(t)$
$(\hat{\imath}, \hat{o}) = \mathrm{L}(\breve{\imath}, \check{\imath})$
uses
Algorithm 5.1: Automatic configuration
provides timing information for
Definition 4.1: IPEP
M
ROL
enables
uses
configures the execution of
configures
executes on
Definition 2.2: CODESYS
$\dot{\mathbf{x}}_i(t) = \mathbf{f}_i(\mathbf{x}_i(t), \mathbf{u}_i(t))$
$\mathbf{y}_i(t) = \mathbf{g}_i(\mathbf{x}_i(t), \mathbf{u}_i(t))$
$\mathbf{x}_i(0) = \mathbf{x}_{0i}$
$u_{\breve{\imath}\check{\imath}}(t) = y_{\hat{\imath}\hat{o}}(t)$
$(\hat{\imath}, \hat{o}) = \mathrm{L}(\breve{\imath}, \check{\imath})$
wraps
Definition 2.14: CSN
M
ROL
wraps
Defintion 3.2: SDFG
M
ROL
is model solved in
is model of computation for
Non-iterative co-simulation

Figure 1.3: The above figure shows a subset of definitions, theorems, and algorithms that provide insight into the work done in this thesis. Algorithm 5.1 is the main contribution of this thesis.

Figure 1.3 shows a subset of definitions, theorems and algorithms that can give a short summary of work done in this thesis. Since definitions, theorems and algorithms are numbered with the chapter number, the above figure also gives an overview of the chapters. Throughout the thesis two systems are used as examples to demonstrate different aspects of the co-simulation. The systems presented are a simple control loop and a two-mass oscillator. The code used in the examples presented in this thesis can be found at [44].

Chapter 2 describes the model of the system under development that is adopted in this thesis. It is assumed that the system can be modeled with CODE system (CODESYS) introduced by Definition 2.2. The co-simulation defect is divided into three parts, and it is shown that the co-simulation error is limited when these three defects are limited (Theorem 2.12). In the same chapter, co-simulation network (CSN) is presented (Definition 2.14). A CSN contains a partial solution from CODESYS. A CSN has a solver for each CSL, as shown in Figure 1.1. Chapter 2 does not contain a CSM specification.

Chapter 3 introduces synchronous data flow (SDF) as the model of computation (MOC) for non-iterative co-simulation It is proposed to wrap a CSN with an SDFG to model the execution of the non-iterative co-simulation. This is a determinate MOC with a large amount of research into scheduling algorithms. This allows this research to be reused to run a non-iterative co-simulation. The proposed wrapping procedure (Definition 3.25) is the specification for constructing a non-iterative CSM. Theorems proving the validity of the proposed wrapper are included in the same chapter.

Chapter 4 provides an ideal execution platform (Definition 4.1) for analyzing the real-time capabilities of the proposed formalism. This analysis suggests a method for configuring the master with which a non-iterative co-simulation can be executed in real time (Definition 4.8). This analysis gives an estimate whether a HIL simulation is possible with the given CSLs (Theorem 4.17). The same chapter shows MBD using examples.

Chapter 5 shows how an estimate of the co-simulation defect can be calculated. This method provides a quality criterion for evaluating the co-simulation quality. Such a criterion can be used to determine the initial values of SDFG tokens and step sizes of simulators. Theorems 2.12, 5.4 and 2.17 justify the automatic procedure shown in Algorithm 5.1 for the configuration of the non-iterative co-simulation. This algorithm is the end result of this thesis. It is included in the chapter on defect calculation because it depends heavily on the results of the defect analysis.

The last chapter of this thesis lists contributions and topics for future work.

Chapter 2

System model

According to [45], a model is "something that represents another thing, either as a physical object that is usually smaller than the real object, or as a simple description that can be used in calculations". In this section, coupled ordinary differential equations (CODE) are introduced as a model of a complex system (Section 2.1). When engineers use such a model, their main concern is to predict the behavior of the technical system under development. The behavior is represented with signal values obtained by solving the model. In this thesis, one of the main concerns is to make sure that the prediction of the behavior by means of co-simulation is reliable, i.e. the co-simulation should deliver results with a small numerical error. From this point of view, CODE is a model that specifies how the numerical error of the co-simulation behaves. An error analysis of the co-simulation is shown in Section 2.2.

Coupled ordinary differential equations (CODE) are differential-algebraic equations of index 1 [46]. Compared to standard notation, algebraic equations are divided into output and connection equations. This separation of algebraic equations was introduced because the solver of a subsystem is responsible for solving its output equations and the co-simulation master is responsible for solving connection equations. The error analysis presented in this chapter shows how the numerical error depends on the defect of a numerical solution [41, 47]. The numerical defect analysis highlights that the responsibility for controlling numerical defect during a co-simulation is distributed. Distributed numerical responsibility for errors is a natural consequence of distributed modeling and simulation. The presented error analysis justifies the measurement of the co-simulation quality introduced in Chapter 5.

A CODE system (CODESYS) consists of state, output and connection equations that model a complex system. A CODE system (CODESYS) describes a complex system at the equation level [7]. This chapter also specifies the behavioral description of a system according to the FMI standard [5] for co-simulation (Section 2.3). In practice, a subsystem of the complex system is modeled in a modeling tool that is suitable for the technical domain of the subsystem. The modeling tools can provide a way to export the package that contains the model and its solver. According to the FMI standard this package is a Functional Mock-up Unit (FMU) for co-simulation (Figure 1.1). The list of tools that are able to export an FMU for co-simulation is maintained at [6]. Such a deployment enables an engineer to select a solver which is suitable for solving the model in the respective technical domain.

An FMU contains `modelDescription.xml` and files that provide the implementation of the CSL. The file `modelDescription.xml` describes the capabilities, ports and parameters of the FMU. The files that provide the implementation of the slave are either shared libraries and/or the source code. An FMU can have multiple shared libraries for multiple operating systems. This can enable the use of an FMU for office simulation under Windows [48] and for HIL simulation under INtime [49]. An FMU does not have to contain the source code. This enables the distribution of simulation models while protecting intellectual property. This property is important for commercial distributions of simulation models. This chapter was created to connect the research in this thesis with a standard used in practice.

It should be noted that there exists an FMU for model exchange. The difference between co-simulation and model exchange is explained in [7]. An FMU for co-simulation can be coupled to other FMUs for co-simulation at the signal level. An FMU for model exchange can be coupled to other FMUs for model exchange at the equation level. An example of coupling at the equation level is shown in Section 2.1. The result of coupling a CODESYS (2.4) at the equation level is (2.13). After such a coupling, a standard ODE or DAE solver [46, 50] can be used to find a numerical solution. This thesis focuses on co-simulation, i.e. on the signal level coupling. Although this work does not focus on model exchange, it introduces CODESYS as a specification for coupling at the equation level. Such a specification is introduced to enable co-simulation error analysis.

2.1 CODE system

This thesis assumes that the simulated system can be modeled by coupled ordinary differential equations (CODE). A formal specification of the CODE system (CODESYS) is introduced in Definition 2.2. This section describes the assumptions that are sufficient for the analytical solution of a CODESYS to exist and can be uniquely determined (Theorem 2.5). The proof closely follows the text about differentiable-algebraic equations of index 1 found in [46]. The error analysis in the next section is based on the same assumptions.

This section presents two examples that are used throughout the thesis. Example 2.6 is introduced to show how co-simulation can be used to design a simple controller. Examples of the design of more complex control systems can be found in [11, 51, 52]. Example 2.7 is introduced to demonstrate the co-simulation of a simple mechanical system. The presented two-mass oscillator is the system that is often used for benchmarking co-simulation master algorithms [15, 31, 39, 53, 54, 55]. According to [2] the average reported co-simulation scenario in the surveyed literature includes only two slaves. The two-mass oscillator system was partitioned into three subsystems to emphasize that the methods presented in the thesis generalize to any number of slaves.

Definition 2.1 (CODE subsystem)**.** A CODE subsystem (CODESUB) is a tuple

$$M = (\mathbf{f}, \mathbf{g}, \mathbf{x}_0) \tag{2.1}$$

where

- $\mathbf{f} : \mathbb{R}^{N_\mathbf{x}} \to \mathbb{R}^{N_\mathbf{x}}$ is the state transition function,
- $\mathbf{g} : \mathbb{R}^{N_\mathbf{x}} \times \mathbb{R}^{N_\mathbf{u}} \to \mathbb{R}^{N_\mathbf{y}}$ is the output function,
- and $\mathbf{x}_0 \in \mathbb{R}^{N_\mathbf{x}}$ is the initial state of the CODESUB.

A CODESUB represents the system of equations

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t)) \tag{2.2a}$$

$$\mathbf{y}(t) = \mathbf{g}(\mathbf{x}(t), \mathbf{u}(t)) \tag{2.2b}$$

$$\mathbf{x}(0) = \mathbf{x}_0 \tag{2.2c}$$

where

- $\mathbf{x} : \mathbb{R}_{\geqslant 0} \to \mathbb{R}^{N_\mathbf{x}}$ is the state signal,
- $\mathbf{y} : \mathbb{R}_{\geqslant 0} \to \mathbb{R}^{N_\mathbf{y}}$ is the output signal,
- and $\mathbf{u} : \mathbb{R}_{\geqslant 0} \to \mathbb{R}^{N_\mathbf{u}}$ is the input signal.

Definition 2.2 (CODE system)**.** A CODE system (CODESYS) is a tuple

$$S = (I_\mathrm{M}, \mathrm{M}, \mathrm{L}) \tag{2.3}$$

where

- $\mathcal{M}$ is the set of all possible CODE subsystems (Definition 2.1),
- $I_\mathrm{M} \subset \mathbb{N}$ is the set of CODESUB labels,
- $\mathrm{M} :\ I_\mathrm{M} \to \mathcal{M}$ is the function for selecting CODESUBs based on the label,
- and $\mathrm{L} :\ \mathbb{N} \times \mathbb{N} \to \mathbb{N} \times \mathbb{N}$ is the connection function.

A CODESYS is given by its CODESUBs

$$\dot{\mathbf{x}}_i(t) = \mathbf{f}_i(\mathbf{x}_i(t), \mathbf{u}_i(t)) \tag{2.4a}$$

$$\mathbf{y}_i(t) = \mathbf{g}_i(\mathbf{x}_i(t), \mathbf{u}_i(t)) \tag{2.4b}$$

$$\mathbf{x}_i(0) = \mathbf{x}_{0i} \tag{2.4c}$$

and the connection function $\mathrm{L} : \mathbb{N} \times \mathbb{N} \to \mathbb{N} \times \mathbb{N}$

$$u_{\breve{i}\breve{\imath}}(t) = y_{\hat{i}\hat{o}}(t), \quad (\hat{i},\hat{o}) = \mathrm{L}(\breve{i},\breve{\imath}) \tag{2.5}$$

where

- $i, \breve{i}, \hat{i} \in \{1, 2, \ldots N_{\mathrm{M}}\}$ are indices of subsystems,
- $\breve{\imath} \in \mathbb{N}$ is the index of an input port,
- and $\hat{o} \in \mathbb{N}$ is the index of an output port.

The input signal of a CODESUB is a vector signal

$$\mathbf{u}(t) = \begin{bmatrix} u_1(t) & u_2(t) & \ldots & u_\imath(t) & \ldots & u_{N_\mathbf{u}}(t) \end{bmatrix}^T \tag{2.6}$$

The index of an input port $\imath$ is the index of a vector element. The signal at the input port indexed $\imath$ is $u_\imath : \mathbb{R}_{\geqslant 0} \to \mathbb{R}$. The analogy holds for output signals.

The signals of all CODESUBs can be further grouped into column vector signals to simplify the notation

$$\begin{aligned} \mathbf{X}^T(t) &= \begin{bmatrix} \mathbf{x}_1{}^T(t) & \mathbf{x}_2{}^T(t) & \cdots & \mathbf{x}_N{}^T(t) \end{bmatrix} \\ \mathbf{Y}^T(t) &= \begin{bmatrix} \mathbf{y}_1{}^T(t) & \mathbf{y}_2{}^T(t) & \cdots & \mathbf{y}_N{}^T(t) \end{bmatrix} \\ \mathbf{U}^T(t) &= \begin{bmatrix} \mathbf{u}_1{}^T(t) & \mathbf{u}_2{}^T(t) & \cdots & \mathbf{u}_N{}^T(t) \end{bmatrix} \end{aligned} \tag{2.7}$$

Then the equation (2.4) can be rewritten in

$$\dot{\mathbf{X}}(t) = \mathbf{F}(\mathbf{X}(t), \mathbf{U}(t)) \tag{2.8a}$$

$$\mathbf{Y}(t) = \mathbf{G}(\mathbf{X}(t), \mathbf{U}(t)) \tag{2.8b}$$

$$\mathbf{U}(t) = \mathbf{L}\mathbf{Y}(t) \tag{2.8c}$$

$$\mathbf{X}(0) = \mathbf{X}_0 \tag{2.8d}$$

where the connection matrix $\mathbf{L} = [l_{ab}] \in \{0,1\}^{N_\mathbf{U} \times N_\mathbf{Y}}$ is formed element by element

$$l_{ab} = \begin{cases} 1, & (\hat{i},\hat{o}) = \mathrm{L}(\breve{i},\breve{\imath}), \quad a = (\breve{i}-1)N_\mathbf{U} + \breve{\imath}, \quad b = (\hat{i}-1)N_\mathbf{Y} + \hat{o} \\ 0, & otherwise \end{cases} \tag{2.9}$$

Definition 2.3 (Lipschitz continuity). A function $\mathbf{F} : \mathbb{R}^{N_\mathbf{X}} \times \mathbb{R}^{N_\mathbf{U}} \to \mathbb{R}^{N_\mathbf{X}}$ is said to be Lipschitz continuous if there exist constant $K_\mathbf{F} > 0$ such that for all $\boldsymbol{\chi}_1, \boldsymbol{\chi}_2 \in \mathbb{R}^{N_\mathbf{X}}$, $\boldsymbol{\upsilon}_1, \boldsymbol{\upsilon}_2 \in \mathbb{R}^{N_\mathbf{U}}$:

$$\left\|\mathbf{F}(\boldsymbol{\chi}_2, \boldsymbol{\upsilon}_2) - \mathbf{F}(\boldsymbol{\chi}_1, \boldsymbol{\upsilon}_1)\right\| \leqslant K_\mathbf{F}\left\|\boldsymbol{\chi}_2 - \boldsymbol{\chi}_1\right\| + K_\mathbf{F}\left\|\boldsymbol{\upsilon}_2 - \boldsymbol{\upsilon}_1\right\| \tag{2.10}$$

The constant $K_{\mathbf{F}}$ is called the Lipschitz constant of the function $\mathbf{F}$.

Lemma 2.4. *If the matrix* $\mathbf{I}-\mathbf{L}\frac{\partial \mathbf{G}(\boldsymbol{\chi},\boldsymbol{\upsilon})}{\partial \boldsymbol{\upsilon}}$ *exists and is invertible, there exists a differentiable function* $\mathbf{H}:\mathbb{R}^{N_{\mathbf{X}}}\to\mathbb{R}^{N_{\mathbf{U}}}$ *where*

$$\mathbf{U}(t)=\mathbf{H}\big(\mathbf{X}(t)\big) \tag{2.11}$$

Proof. The equations (2.8b) and (2.8c) give

$$\mathbf{U}(t)-\mathbf{L}\mathbf{G}\big(\mathbf{X}(t),\mathbf{U}(t)\big)=\mathbf{0} \tag{2.12}$$

The statement of the lemma is a direct consequence of the implicit function theorem [56]. $\square$

Under the conditions of Lemma 2.4, a CODESYS (Definition 2.2) can be reduced to an ordinary differential equation. This fact makes it possible to formulate necessary conditions for the existence and uniqueness of a solution for the CODESYS.

Theorem 2.5 (Existence and Uniqueness)**.** *Suppose the state transition function* $\mathbf{F}$ *is Lipschitz continuous (Definition 2.3) with the Lipschitz constant* $K_{\mathbf{F}}$*. Suppose the matrix* $\mathbf{I}-\mathbf{L}\frac{\partial \mathbf{G}(\boldsymbol{\chi},\boldsymbol{\upsilon})}{\partial \boldsymbol{\upsilon}}$ *exists and is invertible. Then the solution of a CODESYS (Definition 2.2) exists and is unique.*

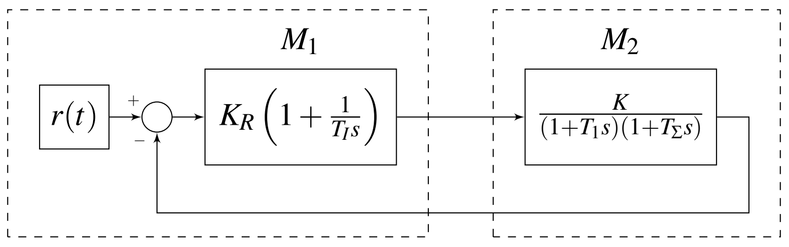
Proof. Lemma 2.4 states that the system (2.8) has an explicit expression for input signals (2.11). The system can be transformed to an ordinary differential equation

$$\dot{\mathbf{X}}(t)=\mathbf{\Phi}\big(\mathbf{x}(t)\big)=\mathbf{\Phi}\big(\mathbf{x}(t),\mathbf{H}\left(\mathbf{X}(t)\right)\big) \tag{2.13}$$

Since $\mathbf{H}$ is differentiable, it is also Lipschitz continuous with the Lipschitz constant $K_{\mathbf{H}}$. The function $\mathbf{F}$ is Lipschitz continuous with the Lipschitz constant

$$K_{\mathbf{\Phi}}=K_{\mathbf{F}}+K_{\mathbf{F}}K_{\mathbf{H}} \tag{2.14}$$

Existence and uniqueness of the solution is a consequence of Picard's theorem for ODEs [57]. $\square$



M_1
M_2
$r(t)$
+
−
$K_R\left(1+\frac{1}{T_I s}\right)$
$\frac{K}{(1+T_1 s)(1+T_\Sigma s)}$

Figure 2.1: The CODESYS presented in Example 2.6.

Example 2.6 (Control loop)**.** A simple control loop is a CODESYS (Definition 2.2)

$$S = (I_\mathrm{M}, \mathrm{M}, \mathrm{L}) \tag{2.15}$$

consisting of two interconnected CODESUBs, as shown in Figure 2.1. The set of CODESUB labels is

$$I_\mathrm{M} = \{1, 2\} \tag{2.16}$$

and the function assigning labels to CODESUBs is

$$\mathrm{M}(i) = \begin{cases} M_1, & i = 1 \\ M_2, & i = 2 \end{cases} \tag{2.17}$$

The subsystem

$$M_1 = (\mathbf{f}_1, \mathbf{g}_1, \mathbf{x}_{01}) \tag{2.18}$$

is a PI controller represented by its transfer function in the diagram. Its equations are

$$\begin{aligned}
\dot{\mathbf{x}}_1(t) &= \mathbf{f}_1(\mathbf{x}_1(t), \mathbf{u}_1(t)) = \begin{bmatrix} \dot{x}_{11}(t) \end{bmatrix} = \begin{bmatrix} \frac{1}{T_I} r(t) - \frac{1}{T_I} u_{11}(t) \end{bmatrix} \\
\mathbf{y}_1(t) &= \mathbf{g}_1(\mathbf{x}_1(t), \mathbf{u}_1(t)) = \begin{bmatrix} y_{11}(t) \end{bmatrix} = K_R \begin{bmatrix} x_{11}(t) \end{bmatrix} + K_R \begin{bmatrix} u_{11}(t) \end{bmatrix} = K_R \mathbf{x}_1(t) + K_R \mathbf{u}_1(t) \\
\mathbf{x}_1(0) &= \mathbf{x}_{01} = \begin{bmatrix} x_{11}(0) \end{bmatrix} = 0
\end{aligned} \tag{2.19}$$

The subsystem

$$M_2 = (\mathbf{f}_2, \mathbf{g}_2, \mathbf{x}_{02}) \tag{2.20}$$

is a simple plant represented by its transfer function in the diagram. Its equations are

$$\begin{aligned}
\dot{\mathbf{x}}_2(t) = \mathbf{f}_2(\mathbf{x}_2(t), \mathbf{u}_2(t)) &= \begin{bmatrix} \dot{x}_{21}(t) \\ \dot{x}_{22}(t) \end{bmatrix} = \begin{bmatrix} -\frac{1}{T_1} x_{21}(t) + \frac{1}{T_1} u_{21}(t) \\ \frac{1}{T_\Sigma} x_{21} - \frac{1}{T_\Sigma} x_{22}(t)(t) \end{bmatrix} \\
&= \begin{bmatrix} -\frac{1}{T_1} & 0 \\ \frac{1}{T_\Sigma} & -\frac{1}{T_\Sigma} \end{bmatrix} \mathbf{x}_2(t) + \begin{bmatrix} -\frac{1}{T_1} \\ 0 \end{bmatrix} \mathbf{u}_2(t) \\
\mathbf{y}_2(t) = \mathbf{g}_2(\mathbf{x}_2(t), \mathbf{u}_2(t)) &= \begin{bmatrix} y_{21}(t) \end{bmatrix} = \begin{bmatrix} K x_{22}(t) \end{bmatrix} = \begin{bmatrix} K & 0 \end{bmatrix} \mathbf{x}_2(t) \\
\mathbf{x}_2(0) = \mathbf{x}_{02} &= \begin{bmatrix} x_{21}(0) \\ x_{22}(0) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}
\end{aligned} \tag{2.21}$$

The connections are given with the following equation

$$u_{21}(t) = y_{11}(t), \quad u_{11}(t) = y_{21}(t) \tag{2.22}$$

or alternatively, with the following connection function

$$\mathrm{L}(i,\iota) = \begin{cases} (1,1), & (i,\iota) = (2,1) \\ (2,1), & (i,\iota) = (1,1) \end{cases} \tag{2.23}$$

The parameters of the example system are chosen as

$$\begin{gathered} K = 1, \quad T_1 = 5, \quad T_\Sigma = 1 \\ T_I = T_1 = 5, \quad K_R = \frac{T_1}{2KT_\Sigma} = 2.5 \end{gathered} \tag{2.24}$$

The previous examples show that a simple control loop can be described by a system of coupled ordinary differential equations (2.4). This example can be generalized for the simulation of a large number of control systems [58].

Example 2.7 (Two mass oscillator)**.** A two-mass oscillator is a CODESYS (Definition 2.2)

$$S = (I_\mathrm{M}, \mathrm{M}, \mathrm{L}) \tag{2.25}$$

consisting of three interconnected CODESUBs M_1, M_2 and M_3, as shown in Figure 2.2. The set of CODESUB labels is

$$I_\mathrm{M} = \{1,2,3\} \tag{2.26}$$

and the function assigning labels to CODESUBs is

$$\mathrm{M}(i) = \begin{cases} M_1, & i = 1 \\ M_2, & i = 2 \\ M_3, & i = 3 \end{cases} \tag{2.27}$$

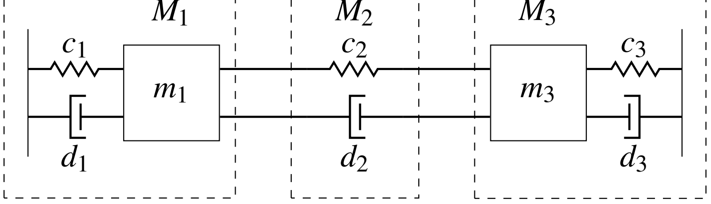


Figure 2.2: The CODESYS presented in Example 2.7.

The subsystem

$$M_1 = (\mathbf{f}_1, \mathbf{g}_1, \mathbf{x}_{01}) \tag{2.28}$$

is the oscillator connected to the left wall

$$\begin{aligned}
\dot{\mathbf{x}}_1(t) = \mathbf{f}_1(\mathbf{x}_1(t), \mathbf{u}_1(t)) &= \begin{bmatrix} \dot{x}_{11}(t) \\ \dot{x}_{12}(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{c_1}{m_1} & -\frac{d_1}{m_1} \end{bmatrix} \begin{bmatrix} x_{11}(t) \\ x_{12}(t) \end{bmatrix} + \begin{bmatrix} \frac{1}{m_1} \end{bmatrix} \begin{bmatrix} u_{11}(t) \end{bmatrix} \\
\mathbf{y}_1(t) = \mathbf{g}_1(\mathbf{x}_1(t), \mathbf{u}_1(t)) &= \begin{bmatrix} y_{11}(t) \end{bmatrix} = \begin{bmatrix} x_{12}(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \end{bmatrix} \mathbf{x}_1(t) \\
\mathbf{x}_1(0) = \mathbf{x}_{01} &= \begin{bmatrix} x_{11}(0) \\ x_{12}(0) \end{bmatrix} = \begin{bmatrix} 0.1 \\ 0.1 \end{bmatrix}
\end{aligned} \tag{2.29}$$

The subsystem

$$M_2 = (\mathbf{f}_2, \mathbf{g}_2, \mathbf{x}_{02}) \tag{2.30}$$

is the coupling element between the two oscillators

$$\begin{aligned}
\dot{\mathbf{x}}_2(t) = \mathbf{u}_2(\mathbf{x}_2(t), \mathbf{u}_2(t)) &= \begin{bmatrix} \dot{x}_{21}(t) \end{bmatrix} = \begin{bmatrix} u_{22}(t) - u_{21}(t) \end{bmatrix} = \begin{bmatrix} -1 & 1 \end{bmatrix} \mathbf{u}_2(t) \\
\mathbf{y}_2(t) = \mathbf{g}_2(\mathbf{x}_2(t), \mathbf{u}_2(t)) &= \begin{bmatrix} y_{21}(t) \\ y_{22}(t) \end{bmatrix} = \begin{bmatrix} -c_2 & 0 \\ 0 & c_2 \end{bmatrix} \mathbf{x}_2(t) + \begin{bmatrix} d_2 & -d_2 \\ -d_2 & d_2 \end{bmatrix} \mathbf{u}_2(t) \\
\mathbf{x}_2(0) = \mathbf{x}_{02} &= \begin{bmatrix} x_{21}(0) \end{bmatrix} = \begin{bmatrix} 1 \end{bmatrix}
\end{aligned} \tag{2.31}$$

The subsystem

$$M_3 = (\mathbf{f}_3, \mathbf{g}_3, \mathbf{x}_{03}) \tag{2.32}$$

is the oscillator connected to the right wall

$$\begin{aligned}
\dot{\mathbf{x}}_3(t) = \mathbf{f}_3(\mathbf{x}_3(t), \mathbf{u}_3(t)) &= \begin{bmatrix} \dot{x}_{31}(t) \\ \dot{x}_{32}(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{c_3}{m_3} & -\frac{d_3}{m_3} \end{bmatrix} \begin{bmatrix} x_{31}(t) \\ x_{32}(t) \end{bmatrix} + \begin{bmatrix} \frac{1}{m_2} \end{bmatrix} \begin{bmatrix} u_{31}(t) \end{bmatrix} \\
\mathbf{y}_3(t) = \begin{bmatrix} y_{31}(t) \end{bmatrix} &= \begin{bmatrix} x_{32}(t) \end{bmatrix} = \mathbf{g}_3(\mathbf{x}_3(t), \mathbf{u}_3(t)) = \begin{bmatrix} 0 & 1 \end{bmatrix} \mathbf{x}_3(t) \\
\mathbf{x}_3(0) = \mathbf{x}_{03} &= \begin{bmatrix} x_{31}(0) \\ x_{32}(0) \end{bmatrix} = \begin{bmatrix} 0.2 \\ 0.1 \end{bmatrix}
\end{aligned} \tag{2.33}$$

The connections are given with the following equation

$$\begin{aligned} u_{21}(t) &= y_{11}(t), \quad u_{11}(t) = y_{21}(t) \\ u_{31}(t) &= y_{22}(t), \quad u_{22}(t) = y_{31}(t) \end{aligned} \tag{2.34}$$

or alternatively, with the following connection function

$$\mathrm{L}(i,\imath) = \begin{cases} (1,1), & (i,\imath) = (2,1) \\ (2,1), & (i,\imath) = (1,1) \\ (3,1), & (i,\imath) = (2,2) \\ (2,2), & (i,\imath) = (3,1) \end{cases} \tag{2.35}$$

The parameters of the example system are chosen as

$$\begin{aligned} & m_1 = 10, \quad c_1 = 1, \quad d_1 = 1, \quad c_2 = 1, \quad d_2 = 2 \\ & m_3 = 10, \quad c_3 = 1, \quad d_3 = 2 \end{aligned} \tag{2.36}$$

The previous example shows how a mechanical system can be modeled by a CODESYS. This example is often used for co-simulation benchmarking. One of the reasons for this may be that it can easily be generalized to any system that can be modeled by a bond-graph [59].

2.2 Numerical solution

This section contains an analysis of a numerical solution for CODESYS (Definition 2.2). In the previous section, Theorem 2.5 showed the conditions under which the analytical solution of a CODESYS is unique. Theorem 2.12 shows that the error of the numerical solution (Definition 2.10) is limited by its defect (Definition 2.8) under the same conditions. The influence of the numerical defect on the numerical error was investigated for differential-algebraic equations [41, 47]. This section follows similar steps when analyzing the error of a CODESYS (Definition 2.2).

Definition 2.8 (Numerical solution)**.** The numerical solution of a CODESYS (Definition 2.2) satisfies the following equations

$$\dot{\widetilde{\mathbf{x}}}_i(t) = \mathbf{f}_i\big(\widetilde{\mathbf{x}}_i(t), \widetilde{\mathbf{u}}_i(t)\big) + \delta\widetilde{\mathbf{x}}_i(t) \tag{2.37a}$$

$$\widetilde{\mathbf{y}}_i(t) = \mathbf{g}_i\big(\widetilde{\mathbf{x}}_i(t), \widetilde{\mathbf{u}}_i(t)\big) + \delta\widetilde{\mathbf{y}}_i(t) \tag{2.37b}$$

$$\widetilde{\mathbf{x}}_i(0) = \mathbf{x}_{0i} \tag{2.37c}$$

$$\widetilde{u}_{\breve{\imath}\breve{\imath}}(t) = \widetilde{y}_{\hat{\imath}\hat{o}}(t) + \delta\widetilde{u}_{\breve{\imath}\breve{\imath}}(t), \quad (\hat{\imath},\hat{o}) = \mathrm{L}\left(\breve{\imath},\breve{\imath}\right) \tag{2.37d}$$

where

- $\widetilde{\mathbf{x}}_i : \mathbb{R}_{\geqslant 0} \to \mathbb{R}^{N_{\mathbf{x}_i}}$ is the numerical solution for the state signal,
- $\delta\widetilde{\mathbf{x}}_i : \mathbb{R}_{\geqslant 0} \to \mathbb{R}^{N_{\mathbf{x}_i}}$ is the integration defect,
- $\widetilde{\mathbf{y}}_i : \mathbb{R}_{\geqslant 0} \to \mathbb{R}^{N_{\mathbf{y}_i}}$ is the numerical solution for the output signal,
- $\delta\widetilde{\mathbf{y}}_i : \mathbb{R}_{\geqslant 0} \to \mathbb{R}^{N_{\mathbf{y}_i}}$ is the output defect,
- $\widetilde{u}_{\widetilde{\imath}} : \mathbb{R}_{\geqslant 0} \to \mathbb{R}$ is the numerical solution found for the input signal,
- and $\delta\widetilde{u}_{\widetilde{\imath}} : \mathbb{R}_{\geqslant 0} \to \mathbb{R}$ is the connection defect.

Lemma 2.9. *If the matrix* $\mathbf{I} - \mathbf{L}\frac{\partial \mathbf{G}(\boldsymbol{\chi},\boldsymbol{\upsilon})}{\partial \boldsymbol{\upsilon}}$ *exists and is invertible then there exists a differentiable function* $\widetilde{\mathbf{H}} : \mathbb{R}^{N_\mathbf{X}} \times \mathbb{R}^{N_\mathbf{U}} \times \mathbb{R}^{N_\mathbf{Y}} \to \mathbb{R}^{N_\mathbf{U}}$ *where*

$$\widetilde{\mathbf{U}}(t) = \widetilde{\mathbf{H}}\left(\widetilde{\mathbf{X}}(t), \delta\widetilde{\mathbf{Y}}(t), \delta\widetilde{\mathbf{U}}(t)\right) \tag{2.38}$$

Proof. The output (2.37b) and connection equations (2.37d) of the numerical solution can be coupled

$$\widetilde{\mathbf{U}}(t) = \mathbf{L}\mathbf{G}\left(\widetilde{\mathbf{X}}(t), \widetilde{\mathbf{U}}(t)\right) + \mathbf{L}\delta\widetilde{\mathbf{Y}}(t) + \delta\widetilde{\mathbf{U}}(t) \tag{2.39}$$

The statement of the lemma is a direct consequence of the implicit function theorem [56]. □

The statements of Lemma 2.4 and Lemma 2.9 are analogous and it can be shown that

$$\mathbf{H}(\mathbf{X}(t)) = \widetilde{\mathbf{H}}(\mathbf{X}(t), \mathbf{0}, \mathbf{0}) \tag{2.40}$$

Definition 2.10 (Co-simulation error)**.** Suppose a numerical algorithm is solving a CODESYS (Definition 2.2) and produces a numerical solution (Definition 2.8). The integration error of the numerical solution is the difference between the numerical (2.37a) and the analytic solution (2.4a) found for the state signals

$$\Delta\widetilde{\mathbf{X}}(t) = \widetilde{\mathbf{X}}(t) - \mathbf{X}(t) \tag{2.41}$$

The output error of the numerical solution is the difference between the numerical (2.37b) and the analytic solution (2.4b) found for the output signals

$$\Delta\widetilde{\mathbf{Y}}(t) = \widetilde{\mathbf{Y}}(t) - \mathbf{Y}(t) \tag{2.42}$$

The input error of the numerical solution is the difference between the numerical (2.37d) and the analytic solution (2.5) found for the input signals

$$\Delta\widetilde{\mathbf{U}}(t) = \widetilde{\mathbf{U}}(t) - \mathbf{U}(t) \tag{2.43}$$

Lemma 2.11 (A limited defect ⇒ a limited integration error). *If the assumptions of Theorem 2.5 hold then the size of the error satisfies*

$$\|\Delta\widetilde{\mathbf{X}}(t)\| \leqslant \|\Delta\widetilde{\mathbf{X}}(t_{\kappa-1})\| + \int\limits_{t_{\kappa-1}}^{t} K_{\mathbf{F}}\|\Delta\widetilde{\mathbf{X}}(\tau)\| + \delta^{(t_{\kappa-1},t]}\,\mathrm{d}\tau \tag{2.44}$$

where

$$\delta^{(t_{\kappa-1},t]} = \max_{t_{\kappa-1}\leqslant\tau\leqslant t}\left(K_{\mathbf{F}}K_{\widetilde{\mathbf{H}}}\|\delta\widetilde{\mathbf{Y}}(\tau)\| + K_{\mathbf{F}}K_{\widetilde{\mathbf{H}}}\|\delta\widetilde{\mathbf{U}}(\tau)\| + \|\delta\widetilde{\mathbf{X}}(\tau)\|\right) \tag{2.45}$$

Proof. The Lemma 2.9 holds under the assumptions of the Theorem 2.5. Since $\widetilde{\mathbf{H}}$ is differentiable, it is also Lipschitz continuous with the Lipschitz constant $K_{\widetilde{\mathbf{H}}}$. The expression (2.38) and the application of the norm lead to the following inequality

$$\|\Delta\widetilde{\mathbf{U}}(t)\| \leqslant K_{\widetilde{\mathbf{H}}}\|\Delta\widetilde{\mathbf{X}}(t)\| + K_{\widetilde{\mathbf{H}}}\|\delta\widetilde{\mathbf{Y}}(t)\| + K_{\widetilde{\mathbf{H}}}\|\delta\widetilde{\mathbf{U}}(t)\| \tag{2.46}$$

The integration error expression (2.41), the integration of the state equations (2.4a) and (2.37a) result in

$$\Delta\widetilde{\mathbf{X}}(t) = \Delta\widetilde{\mathbf{X}}(t_{\kappa-1}) + \int\limits_{t_{\kappa-1}}^{t}\left[\mathbf{F}\big(\widetilde{\mathbf{X}}(\tau),\widetilde{\mathbf{U}}(\tau)\big) - \mathbf{F}\big(\mathbf{X}(\tau),\mathbf{U}(\tau)\big) + \delta\widetilde{\mathbf{X}}(\tau)\right]\,\mathrm{d}\tau \tag{2.47}$$

After the application of the norm and the Lipschitz continuity, the following inequality is obtained

$$\|\Delta\widetilde{\mathbf{X}}(t)\| \leqslant \|\Delta\widetilde{\mathbf{X}}(t_{\kappa-1})\| + \int\limits_{t_{\kappa-1}}^{t}\left[K_{\mathbf{F}}\|\Delta\widetilde{\mathbf{X}}(\tau)\| + K_{\mathbf{F}}\|\Delta\widetilde{\mathbf{U}}(\tau)\| + \|\delta\widetilde{\mathbf{X}}(\tau)\|\right]\,\mathrm{d}\tau \tag{2.48}$$

The statement of the lemma is proven by (2.45), (2.46) and (2.48). □

Theorem 2.12 (Error bounds). *Suppose the assumptions of Theorem 2.5 hold and the numerical solution is continuous in every sub-interval* $(t_{\kappa-1},t_\kappa]$. *Then the integration error satisfies the inequality*

$$\|\Delta\widetilde{\mathbf{X}}(t)\| \leqslant e^{K_{\mathbf{F}}(t-t_0)}\|\Delta\widetilde{\mathbf{X}}(t_0)\| + \frac{1}{K_{\mathbf{F}}}\left(e^{K_{\mathbf{F}}(t-t_0)} - 1\right)\delta^{(t_0,t]} \tag{2.49}$$

the input error satisfies the inequality

$$\begin{aligned}\|\Delta\widetilde{\mathbf{U}}(t)\| \leqslant & K_{\widetilde{\mathbf{H}}}e^{K_{\mathbf{F}}(t-t_0)}\|\Delta\widetilde{\mathbf{X}}(t_0)\| + \frac{K_{\widetilde{\mathbf{H}}}}{K_{\mathbf{F}}}\left(e^{K_{\mathbf{F}}(t-t_0)} - 1\right)\delta^{(t_0,t]} \\ & + K_{\widetilde{\mathbf{H}}}\|\delta\widetilde{\mathbf{Y}}(t)\| + K_{\widetilde{\mathbf{H}}}\|\delta\widetilde{\mathbf{U}}(t)\|\end{aligned} \tag{2.50}$$

and the output error satisfies the inequality

$$\begin{aligned}\|\Delta\widetilde{\mathbf{Y}}(t)\| \leqslant & K_{\mathbf{G}}\left(1+K_{\widetilde{\mathbf{H}}}\right)e^{K_{\mathbf{F}}(t-t_0)}\|\Delta\widetilde{\mathbf{X}}(t_0)\| \\ & +\frac{K_{\mathbf{G}}}{K_{\mathbf{F}}}\left(1+K_{\widetilde{\mathbf{H}}}\right)\left(e^{K_{\mathbf{F}}(t-t_0)}-1\right)\delta^{(t_0,t]} \\ & +\left(1+K_{\mathbf{G}}K_{\widetilde{\mathbf{H}}}\right)\|\delta\widetilde{\mathbf{Y}}(t)\|+K_{\mathbf{G}}K_{\widetilde{\mathbf{H}}}\|\delta\widetilde{\mathbf{U}}(t)\|\end{aligned} \tag{2.51}$$

Proof. The inequality (2.44) holds as a consequence of Lemma 2.11. The Gronwall lemma (Theorem 6 in [60]) shows that for a continuous numerical solution the inequality (2.44) implies

$$\|\Delta\widetilde{\mathbf{X}}(t)\|+\frac{\delta^{(t_{\kappa-1},t]}}{K_{\mathbf{F}}} \leqslant e^{K_{\mathbf{F}}(t-t_{\kappa-1})}\left(\|\Delta\widetilde{\mathbf{X}}(t_{\kappa-1})\|+\frac{\delta^{(t_{\kappa-1},t]}}{K_{\mathbf{F}}}\right) \tag{2.52}$$

Since $\delta^{(t_{\kappa-1},t]} \leqslant \delta^{(t_0,t]}$ for all $t_\kappa \in [t_0,t]$

$$\|\Delta\widetilde{\mathbf{X}}(t_\kappa)\|+\frac{\delta^{(t_0,t]}}{K_{\mathbf{F}}} \leqslant e^{K_{\mathbf{F}}(t_\kappa-t_{\kappa-1})}\left(\|\Delta\widetilde{\mathbf{X}}(t_{\kappa-1})\|+\frac{\delta^{(t_0,t]}}{K_{\mathbf{F}}}\right) \tag{2.53}$$

The following inequality is obtained by recursion:

$$\|\Delta\widetilde{\mathbf{X}}(t)\|+\frac{\delta^{(t_0,t]}}{K_{\mathbf{F}}} \leqslant e^{K_{\mathbf{F}}(t-t_0)}\left(\|\Delta\widetilde{\mathbf{X}}(t_0)\|+\frac{\delta^{(t_0,t]}}{K_{\mathbf{F}}}\right) \tag{2.54}$$

Since $\mathbf{G}$ is differentiable, it is also Lipschitz continuous with the Lipschitz constant $K_{\mathbf{G}}$. By rearrangement (2.49) is obtained. (2.50) follows from (2.46) and (2.49), while (2.51) follows from (2.37b), (2.49) and (2.50). □

The components of numerical error (2.49)-(2.51) are limited by the error in the initial values $\|\Delta\widetilde{\mathbf{X}}(t_0)\|$, the integration, the output and the connection defect (2.45). If each of the defect components is controlled and kept low, the error that occurs when simulating a CODESYS should be small. It is useful to note that a defect is defined independently for each individual equation (Definition 2.8). In this way, each subsystem can have a different control mechanism for its integration defect [47, 61], while output and connection defects can be monitored using the method described in Chapter 5.

2.3 Co-simulation network specification

This section contains a behavioral description of an FMU (Definition 2.13), a co-simulation slave (CSL) according to the FMI standard [5]. The network of FMUs is also described (Definition 2.14), which forms the basis for the CODE system wrapper (SYSW) presented in the next section.

Definition 2.13 (Co-simulation slave)**.** A co-simulation slave (CSL) according to the FMI 2.0 standard is a tuple

$$F = (I_{\mathrm{U}}, \mathrm{U}, I_{\mathrm{Y}}, \mathrm{Y}, V, v_0, \mathrm{set}, \mathrm{get}, \mathrm{doStep}) \tag{2.55}$$

where

- $I_{\mathrm{U}} = \{1, 2, \ldots, N_{\mathrm{U}}\}$ is the set of input port labels,
- $\mathcal{U}$ is the set of sets of input values,
- $\mathrm{U}: \; I_{\mathrm{U}} \to \mathcal{U}$ is the function that assigns a set of values to the input port,
- $I_{\mathscr{Y}} = \{1, 2, \ldots, N_{\mathrm{Y}}\}$ is the set of output port labels,
- $\mathcal{Y}$ is the set of sets of output values,
- $\mathrm{Y}: \; I_{\mathrm{Y}} \to \mathcal{Y}$ is the function that assigns a set of values to the output port,
- V is a set of internal states,
- $v_0 \in V$ the initial state of the FMU,
- $\mathrm{set}: \; I_{\mathrm{U}} \times V \times \mathcal{Z} \to V$ the function which sets the input values,
- $\mathrm{doStep}: \; \mathbb{R}_{>0} \times V \to V$ the function which calculates the state update,
- $\mathrm{get}: \; I_{\mathrm{Y}} \times V \to \mathcal{Z}$ is the function which returns the output values.

Suppose the slave is updated with the sequence of communication step sizes $h : \mathbb{N} \to \mathbb{R}_{>0}$. The sequence of communication points $t : \mathbb{N}_0 \to \mathbb{R}_{\geqslant 0}$ is determined by the communication step sizes

$$t(n) = t(n-1) + h(n), \quad t(0) = 0 \tag{2.56}$$

Values are assigned to the input ports

$$v^1(n) = \mathrm{set}(1, v(n-1), u(n,1)), \quad v^{\iota}(n) = \mathrm{set}\left(\iota, v^{\iota-1}(n), u(n,\iota)\right) \tag{2.57}$$

The order of the ports has no influence on the final state after the input values have been updated. The internal state of the slave after the n^{th} update is

$$v(n) = \mathrm{doStep}\left(v^{N_{\mathrm{U}}}(n), h(n)\right) \tag{2.58}$$

The input signal reconstructed at the port labeled ι is a piecewise defined signal

$$\underline{\mathrm{u}}_{\iota}(t) = u(n,\iota), \quad t(n-1) < t \leqslant t(n) \tag{2.59}$$

The output signal reconstructed at the port labeled l is a piecewise defined signal

$$\underline{\mathrm{y}}_{l}(t) = y(l,n) = \mathrm{get}(l, v(n)), \quad t(n-1) < t \leqslant t(n) \tag{2.60}$$

The naming of the terms used to define the CSL is similar to the C-API defined in the FMI standard [5]. The comparison of the analogous terms is shown in Table 2.1. The definition above

Definition 2.13	FMI standard [5]
$I_{\mathrm{U}}, I_{\mathrm{Y}}$	value references
$\mathrm{U}(\imath), \mathrm{Y}(l)$	`fmi2Real`, `fmi2Integer`, `fmi2Boolean`, `fmi2String`
V	`fmi2Component`
v_0	`fmi2Instantiate`
set	`fmi2SetReal`, `fmi2SetInteger`, `fmi2SetBoolean`, `fmi2SetString`
get	`fmi2GetReal`, `fmi2GetInteger`, `fmi2GetBoolean`, `fmi2GetString`
doStep	`fmi2DoStep`

Table 2.1: Comparison of analogous terms

does not include the output-input dependency information. This information is necessary for the iterative co-simulation master presented in [29] and can be used to optimize the sequential co-simulation [31]. This choice limits the ability of the proposed co-simulation method to handle hybrid co-simulation requirements [62]. This thesis examines how the co-simulation can be improved in view of the minimal information provided by CSLs.

In this thesis the signal reconstruction is defined by (2.59) and (2.60). The CSLs use ZOH to reconstruct the input and output signals during the state update. Higher order extrapolation techniques are not included in this thesis to ease the notation. However, it should be a straightforward task to generalize the research in this thesis to use them*. The FMI standard [5] provides the means to reconstruct signals using higher order extrapolation techniques.

Definition 2.14 (Co-simulation network)**.** A co-simulation network (CSN) is a tuple

$$C = (I_{\mathrm{F}}, \mathrm{F}, \mathrm{L}) \tag{2.61}$$

where

- $I_{\mathrm{F}} \subset \mathbb{N}$ is the set of CSL labels,
- $\mathcal{F}$ is the set of all possible co-simulation slaves,
- $\mathrm{F}: \ I_{\mathrm{F}} \to \mathcal{F}$ is the function for selecting CSLs based on the label,
- and $\mathrm{L}: \ \mathbb{N} \times \mathbb{N} \to \mathbb{N} \times \mathbb{N}$ is the connection function.

The connection function provides indices of the source slave output port $(\hat{\imath}, \hat{o})$ connected to the

*However, this task can be time consuming.

given destination slave input port $(\breve{\imath}, \breve{\iota})$

$$(\hat{\imath}, \hat{o}) = \mathrm{L}(\breve{\imath}, \breve{\iota}) \tag{2.62}$$

Signals obtained from the input port $\underline{\mathrm{u}}_{\breve{\iota}} : \mathbb{R}_{\geqslant 0} \to \mathcal{Z}$ and the connected output port $\underline{\mathrm{y}}_{\hat{o}} : \mathbb{R}_{\geqslant 0} \to \mathcal{Z}$ should be equal

$$\underline{\mathrm{u}}_{\breve{\iota}}(t) = \underline{\mathrm{y}}_{\hat{o}}(t) \tag{2.63}$$

All of the slaves in the network are weakly connected, i.e. there is an undirected path between any pair of co-simulation slaves.

During the co-simulation, a CSM (Figure 1.1) determines the communication points of each wrapper (2.56) and solves the connection equation (2.63). If a complex system is modeled with a CSN, the solvers and the CSM share responsibility for the numerical solution.

2.4 CODE system wrapper

In practice, a complex system is often described at the signal level. This means that the equations of the CODESYS (2.4) are not available. If subsystem models are available as CSLs, an engineer can use a CSN as a model for the complex system. This approach is useful because it enables distributed modeling. The engineer who creates the CSN can reuse the models exported from various simulation tools [6]. In this way, engineers developing subsystem models can use the most appropriate simulation tools for their domain. This section describes a CODE system wrapper (SYSW) in Definition 2.16. An SYSW is a CSN (Definition 2.14) generated from a CODESYS (Definition 2.2).

In practice, SUBWs are block boxes due to the protection of intellectual property. Examples 2.18 and 2.19 show the influence of the solver on the integration defect. For these reasons, the engineer who creates an SUBW should ensure that a suitable solver is selected. The connection defect is introduced by the CSM (Figure 1.1) when solving (2.63). The co-simulation master (CSM) proposed in this thesis is described in Chapter 3 and its connection defect is analyzed in Chapter 5.

The output defect of an SUBW is analyzed in Theorem 2.17. While responsibility for minimizing integration and connection defect is clear, responsibility for minimizing output defect is shared. The internal SUBW solver solves the output equation. In addition, an SUBW defines the reconstruction of the signal (2.60). However, the output defect depends on the size of the communication step assigned to the slave (Theorem 2.17). The master can estimate the output defect as shown in Chapter 5. In this way, a CSM master can minimize the output defect.

Definition 2.15 (CODE subsystem wrapper)**.** A CODE subsystem wrapper (SUBW) is a CSL (Definition 2.13) that solves a CODESUB

$$\begin{aligned} F &= (I_{\mathrm{U}}, \mathrm{U}, I_{\mathrm{Y}}, \mathrm{Y}, V, v_0, \mathrm{set}, \mathrm{get}, \mathrm{doStep}) \\ &= \mathrm{subsystemToSlave}(M, \mathrm{solver}) \end{aligned} \tag{2.64}$$

where

- $F \in \mathcal{F}$ is a SUBW,
- $M \in \mathcal{M}$ is a CODESUB (Definition 2.1),
- solver : $\mathcal{M} \times V \times \mathbb{R}_{>0} \to V$ is a function solving the CODESUB,
- and subsystemToSlave : $\mathcal{M} \times V^{\mathcal{M} \times V \times \mathbb{R}_{>0}} \to \mathcal{F}$ is a function that constructs the SUBW.

Input port labels of the SUBW are

$$I_{\mathrm{U}} = \{1, 2, \ldots, N_{\mathbf{u}}\} \tag{2.65}$$

The function labeling input port sets of the SUBW is

$$\mathrm{U}(\imath) = \mathbb{R}, \quad \imath \in I_{\mathrm{U}} \tag{2.66}$$

Output port labels of the SUBW are

$$I_{\mathrm{Y}} = \left\{1, 2, \ldots, N_{\mathbf{y}}\right\} \tag{2.67}$$

The function labeling output port set of the SUBW is

$$\mathrm{Y}(o) = \mathbb{R}, \quad o \in I_{\mathrm{Y}} \tag{2.68}$$

The set of internal states is the set of tuples

$$V = \mathbb{R}^{N_{\mathbf{u}}} \times 2^{\mathbb{R}} \tag{2.69}$$

The initial state of the SUBW is

$$v_0 = \big(\mathbf{0}, \mathbf{x}_0\big) \tag{2.70}$$

Input values of the SUBW are set using (2.57) where

$$v^{\imath-1}(n) = \big(u(n,1), \ldots, u(n-1,\imath), \ldots, u(n-1,N_{\mathbf{u}}), \ldots\big) \tag{2.71}$$

and

$$v^{\imath}(n) = \big(u(n,1), \ldots, u(n,\imath), \ldots, u(n-1,N_{\mathbf{u}}), \ldots\big) \tag{2.72}$$

Output values are obtained using (2.60) and the output equation (2.2b) holds for all communication points

$$\mathrm{get}\big(o, v(n)\big) = g_o\Big(\widetilde{\mathbf{x}}\big(t(n)\big), \widetilde{\mathbf{u}}\big(t(n)\big)\Big) \tag{2.73}$$

The state update of the SUBW is

$$\mathrm{doStep}\left(v^{N_\mathrm{U}}(n), h(n)\right) = \mathrm{solver}\left(M, v^{N_\mathrm{U}}(n), h(n)\right) \tag{2.74}$$

A SUBW wraps a CODESUB with the function solver (Figure 1.1). The interface of two different solvers is shown in Figure 1.1. Such a definition enables the implementation of a large number of ODEs [50, 63] and DAE solvers [46]. The work in this thesis is limited to CODESUB to simplify numerical analysis.

Definition 2.16 (CODE system wrapper)**.** A CODE system wrapper (SYSW) is a CSN (Definition 2.14) that solves a CODESYS

$$C = (I_\mathrm{F}, \mathrm{F}, \mathrm{L}) = \mathrm{systemToNetwork}\,(S, \mathrm{subsystemSolvers}) \tag{2.75}$$

where

- $S \in \mathcal{S}$ is a CODESYS (Definition 2.2),
- and subsystemSolvers : $I_\mathrm{M} \to V^{\mathcal{M}\times V^{\mathcal{M}\times V\times\mathbb{R}_{>0}}}$ is the function that assigns a solver to each CODESUB.

The set of CODESUB labels is

$$I_\mathrm{F} = I_\mathrm{M} \tag{2.76}$$

The CSLs are SUBWs (Definition 2.15)

$$\mathrm{F}\,(i) = \mathrm{subsystemToSlave}\big(M, \mathrm{subsystemSolvers}\,(i)\big) \tag{2.77}$$

The connection function L is defined by the CODESYS (2.5).

The previous definition shows how a CSN network can model a complex system. Subsystems are modeled as CSLs (2.77) and coupled with the connection function L. This is an example of coupling at the signal level [7]. The main difference between signal level coupling and equation level coupling (2.13) is that each SUBW contains a solver (Figure 1.1). Such a SUBW is sometimes called a simulator, and coupling at the signal level or co-simulation is also called simulator coupling. In practice, due to legacy software, it is often much easier to couple simulation tools at the signal level than at the equation level. In addition, by using the coupling at the signal level, one of the subsystem models can be replaced by the actual hardware. The co-simulation can be adapted to the HIL simulation and facilitate model-based development (MBD). Hardware components cannot communicate with the rest of the simulation at the equation level.

In practice, engineers treat SUBWs as black boxes and combine them to create an SYSW to use as model of the complex system. The equations of an SYSW and its underlying CODESYS are different due to introduction of solvers. Whenever each SUBW solves its CODESUB equations it leads to integration and output defects

$$\dot{\underline{\mathbf{x}}}_i(t) = \mathbf{f}_i(\underline{\mathbf{x}}_i(t), \underline{\mathbf{u}}_i(t)) + \delta\underline{\mathbf{x}}_i(t) \tag{2.78a}$$

$$\underline{\mathbf{y}}_i(t) = \mathbf{g}_i(\underline{\mathbf{x}}_i(t), \underline{\mathbf{u}}_i(t)) + \delta\underline{\mathbf{y}}_i(t) \tag{2.78b}$$

$$\underline{\mathbf{x}}_i(0) = \mathbf{x}_{0i} \tag{2.78c}$$

The output defect of an SUBW is caused by ZOH extrapolation (2.60). The asymptotic analysis of the output defect is shown in Theorem 2.17. The integration defect caused by a simple solver is calculated in Example 2.18. The integration defect is negligible[†] if the wrapper equations are simple enough to enable the analytical solution (Example 2.19). An SYSW does not introduce a connection defect in the connection equation (2.63). However, a CSM that controls the co-simulation does this. This will be shown in the next chapter (Theorem 5.2).

Theorem 2.17 (Output defect)**.** *Let M be a CODESUB (Definition 2.1) and*

$$F = \text{systemToNetwork}\left(S, \text{subsystemSolvers}\right) \tag{2.79}$$

Suppose the function $\mathbf{g}$ *is continuously differentiable*

$$\underline{\boldsymbol{x}}(t) = \underline{\boldsymbol{x}}\big(t(n)\big) + \left.\frac{\mathrm{d}\underline{\boldsymbol{x}}(\tau)}{\mathrm{d}\tau}\right|_{\tau=t(n)} \big(t - t(n)\big) + \mathscr{O}\big(h^2(n)\big) \tag{2.80}$$

Then the output defect of an SUBW is

$$\delta\underline{\boldsymbol{y}}(t) = \mathscr{O}\big(h(n)\big) \tag{2.81}$$

Proof. From (2.80) and the fact that $\mathbf{g}$ is continuously differentiable, it follows that

$$\mathbf{g}\big(\underline{\mathbf{x}}(t), \underline{\mathbf{u}}(t)\big) = \mathbf{g}\Big(\underline{\mathbf{x}}\big(t(n)\big), \underline{\mathbf{u}}\big(t(n)\big)\Big) + \left.\frac{\mathrm{d}\mathbf{g}\big(\underline{\mathbf{x}}(\tau), \underline{\mathbf{u}}(\tau)\big)}{\mathrm{d}\tau}\right|_{\tau=t(n)} \big(t - t(n)\big) + \mathscr{O}\big(h^2(n)\big) \tag{2.82}$$

The expression (2.81) follows from (2.59), (2.60), (2.73) and (2.78b). □

Theorem 2.17 shows that the output defect can be reduced by the CSM. The CSM can reduce the communication step size to reduce the output defect of an SUBW. An SUBW influences the output defect by choosing the extrapolation method (2.60). For this reason, the responsibility

[†]Floating point arithmetic prevents the integration defect from falling to zero.

for the output defect is shared between the CSM and the SUBW. The next chapters show how the proposed CSM can estimate and reduce the output defect.

Example 2.18 (Control loop)**.** The control loop model S (2.15) shown in Example 2.6 is a CODESYS partitioned into two CODESUBs. This example is a continuation of Example 2.6 and shows how to get an SYSW (Definition 2.16) from the given control loop equations. Let euler : $\mathcal{M} \times V \times \mathbb{R}_{>0} \to V$ be a forward Euler solver

$$v(n) = \text{euler}\,(M, v(n-1), h(n)) \tag{2.83}$$

where the internal state after the integration is

$$v(n) = \Big(\underline{\mathbf{u}}\big(t(n)\big), \underline{\mathbf{x}}\big(t(n)\big)\Big) \tag{2.84}$$

and

$$\underline{\mathbf{x}}\,(t(n)) = \underline{\mathbf{x}}\,(t(n-1)) + h(n)\,\mathbf{f}\big(\underline{\mathbf{x}}\,(t(n-1))\,,\underline{\mathbf{u}}\,(t(n))\big) \tag{2.85}$$

A continuous extension [61] of the state signal produced by the forward Euler solver is

$$\underline{\mathbf{x}}(t) = \underline{\mathbf{x}}\,(t(n-1)) + (t - t(n-1))\;\mathbf{f}\big(\underline{\mathbf{x}}\,(t(n-1))\,,\underline{\mathbf{u}}\,(t(n))\big), \quad t(n)-1 < t \leqslant t(n) \tag{2.86}$$

The integration defect (2.78a) of the forward Euler solver can be calculated using the continuous extension

$$\delta\underline{\mathbf{x}}(t) = \mathscr{O}\big(h(n)\big), \quad t(n)-1 < t \leqslant t(n) \tag{2.87}$$

Both SUBW solvers are forward Euler solvers

$$\text{subsystemSolvers}(i) = \text{euler} \tag{2.88}$$

The control loop SYSW is

$$C = \text{systemToNetwork}(S, \text{subsystemSolvers}) \tag{2.89}$$

Example 2.19 (Two mass oscillator)**.** The two-mass oscillator model S (2.25) shown in Example 2.7 is partitioned into three CODESUBs. This example is a continuation of Example 2.7 and shows how to get an SYSW (Definition 2.16) from the given two-mass oscillator equations. Let analytic : $\mathcal{M} \times V \times \mathbb{R}_{>0} \to V$ be an analytic solver

$$v(n) = \text{analytic}\,(M, v(n-1), h(n)) \tag{2.90}$$

where the internal state after the integration is

$$v(n) = \big(\underline{\mathbf{u}}\,(t(n))\,,\underline{\mathbf{x}}\,(t(n))\big) \tag{2.91}$$

and

$$\underline{\mathbf{x}}\,(t(n)) = \underline{\mathbf{x}}\,(t(n-1)) + \int_{t(n-1)}^{t(n)} \mathbf{f}\big(\underline{\mathbf{x}}\,(\tau)\,,\underline{\mathbf{u}}\,(\tau)\,\big)\mathrm{d}\tau \tag{2.92}$$

In general, an analytical solver cannot be coded. The equations of the subsystem are solved symbolically and the expressions for the analytical solver are hard-coded. Such a solver has no integration defect

$$\delta\underline{\mathbf{x}}(t) = \mathbf{0} \tag{2.93}$$

All SUBW solvers are considered analytical solvers

$$\text{subsystemSolvers}\,(i) = \text{analytic} \tag{2.94}$$

The two-mass oscillator SYSW is

$$C = \text{systemToNetwork}\,(S, \text{subsystemSolvers}) \tag{2.95}$$

Chapter 3

Synchronous data flow

This chapter introduces synchronous data flow (SDF) as the model of computation (MOC) for non-iterative co-simulation. A MOC describes how computation, memory, and communication between components are organized [64]. This makes co-simulation network wrapper (CSW) introduced in Definition 3.25 a specification for modeling a CSM.

An SDF graph (SDFG) (Definition 3.2) is a determinate MOC. This means that any valid execution should produce the same results. Section 3.2 shows a proof for the determinacy of SDFG. Since SDFG is determinate, a CSW can be used as a layer between the CSN and the execution platform. This means that the execution details are abstracted from the CSW. A co-simulation engineer does not need to know the details about the execution platform. This engineer only needs to know if the execution platform can run the generated SDFG.

An SDFG specifies execution on multiple execution platforms. It is the responsibility of the platform developer to ensure that the SDFG is properly executed on the execution platform. The advantage of modeling a non-iterative co-simulation with SDF is that extensive research of planning algorithms is available [35, 36, 65]. It is interesting to note that research on scheduling multi-rate co-simulation on multi-core platforms is available [66]. This thesis proposes to solve the scheduling problem by modeling a non-iterative co-simulation with an SDFG. In this way a large toolbox for solving scheduling problems becomes available. Section 3.3 explains how to execute an SDFG sequentially on a single processor. Chapter 4 shows how to construct an SDFG to enable real-time execution on an idealistic platform.

The authors in [28] state that SDF is useful as a hardware description, but assume that SDF is mainly used for functional description. In [64] the author mentions scheduling and buffer minimization as an synchronous data flow (SDF) application example. An example of the use of SDF for modeling can be found in [27, 37]. In [37] authors show how an SDFG can be packaged as an FMU. This chapter shows how to wrap a CSN as an SDFG. The validity analysis and examples of this wrapper are presented in Section 3.4.

3.1 Synchronous data flow graph

This section describes an SDF graph (Definition 3.2, Example 3.3) and the rules for its execution (Definition 3.7). It contains a number of rules that algorithms must follow, but does not specify the execution algorithm. This enables such a specification to be executed on one or more processors [35, 36].

Definition 3.1 introduces an SDF actor (SDFA). An SDFA is a specification of a repetitive computation. It is used as a building block of an SDF graph (SDFG). In Section 3.4 it is shown how to wrap a CSL with an SDFA.

Definition 3.1 (SDF actor)**.** An SDF actor (SDFA) is a tuple

$$A = (I_{\mathscr{U}}, \mathscr{U}, I_{\mathscr{Y}}, \mathscr{Y}, \mathrm{c}, \mathrm{p}, \mathrm{calculate}) \tag{3.1}$$

where

- $I_{\mathscr{U}} = \{1, 2, \ldots, N_{\mathscr{U}}\}$ is the set of input port labels,
- $\mathcal{U}$ is the set of sets of input values,
- $\mathscr{U} : I_{\mathscr{U}} \to \mathcal{U}$ is the function that assigns a set of values to the input port,
- $I_{\mathscr{Y}} = \{1, 2, \ldots, N_{\mathscr{Y}}\}$ is the set of output port labels,
- $\mathcal{Y}$ is the set of sets of output values,
- $\mathscr{Y} : I_{\mathscr{Y}} \to \mathcal{Y}$ is the function that assigns a set of values to the output port,
- c : $I_{\mathscr{U}} \to \mathbb{N}$ is the function that assigns a consumption rate to the input port,
- p : $I_{\mathscr{Y}} \to \mathbb{N}$ is the function that assigns a production rate to the output port,
- $\mathcal{Z}$ is the set of all possible values,
- and calculate : $\mathcal{Z}^{I_{\mathscr{U}} \times \mathbb{N}} \to \mathcal{Z}^{I_{\mathscr{Y}} \times \mathbb{N}}$ is the function that is called when the SDFA is triggered.

The calculation function

$$\gamma(k') = \mathrm{calculate}\big(\xi(k'')\big) \tag{3.2}$$

is used to obtain the sequence of output tokens $\gamma : \mathbb{N} \to \mathcal{Z}^{I_{\mathscr{Y}} \times \mathbb{N}}$ where $\xi : \mathbb{N} \to \mathcal{Z}^{I_{\mathscr{U}} \times \mathbb{N}}$ is the sequence of input tokens[*]. The values of tokens consumed by the SDFA at the input port $\iota \in I_{\mathscr{U}}$ are

$$\xi\left(k', \iota, \theta\right) \in \mathscr{U}(\iota), \quad 1 \leqslant \theta \leqslant \mathrm{c}(\iota) \tag{3.3}$$

The values of tokens produced by the SDFA at the output port $o \in I_{\mathscr{Y}}$ are

$$\gamma\big(k'', o, \theta\big) \in \mathscr{Y}(o), \quad 1 \leqslant \theta \leqslant \mathrm{p}(l) \tag{3.4}$$

[*]Two notations are used to denote a function. The function $\gamma : \mathbb{N} \times I_{\mathscr{Y}} \times \mathbb{N} \to \mathcal{Z}$ can be denoted as an element of the appropriate function set, i.e. $\gamma \in \mathcal{Z}^{\mathbb{N} \times I_{\mathscr{Y}} \times \mathbb{N}}$.

The input port labeled $N_{\mathscr{U}}$ consumes one token

$$\mathrm{c}(N_{\mathscr{U}}) = 1 \tag{3.5}$$

The output port labeled $N_{\mathscr{Y}}$ produces one token

$$\mathrm{p}(N_{\mathscr{Y}}) = 1 \tag{3.6}$$

In the definition above, the output tokens and input tokens involved in the same calculation (3.2) are indexed with different variables. This highlights that consumption and production of different SDFAs can be interleaved during execution (Definition 3.7). The consumption of the last input port (3.5) and the production of the last output port (3.6) are constrained. The last input port and the last output port are part of the self loop (3.8). The SDFA as defined above is a stateless component. However, the self-loop allows to model a state as part of the SDF graph (SDFG).

SDFAs and SDF buffers (SDFBs) form building blocks of an SDFG (Definition 3.2). An SDFB is a first-in-first-out queue that connects two SDFA ports. SDFBs are implicitly introduced in the next definition and their behavior is described in Definition 3.7.

Definition 3.2 (SDF graph)**.** A SDF graph (SDFG) is a tuple

$$G = \left(I^{\mathrm{A}}, \mathrm{A}, I^{\mathrm{B}}, \mathrm{src}, \mathrm{dst}, \mathrm{d}_0, \omega_0\right) \tag{3.7}$$

where

- $I^{\mathrm{A}} = \{1, 2, \ldots, N_{\mathrm{A}}\}$ is the set of SDFA labels,
- $\mathrm{A} : \ I^{\mathrm{A}} \to \mathcal{A}$ is the function that labels an SDFAs (Definition 3.1),
- $I^{\mathrm{B}} = \left\{1, 2, \ldots, N^{\mathrm{B}}\right\}$ is the set of SDFB labels,
- $\mathrm{src} : \ I^{\mathrm{B}} \to I^{\mathrm{A}}$ is the function that gives the label of the SDFB source actor,
- $\mathrm{dst} : \ I^{\mathrm{B}} \to I^{\mathrm{A}}$ is the function that gives the label of the SDFB destination actor,
- $\mathrm{d}_0 : I^{\mathrm{B}} \to \mathbb{N}_0$ is the function that specifies the number of initial tokens in the SDFBs,
- and $\omega_0 : I^{\mathrm{B}} \to \mathcal{Z}^{\mathbb{N}}$ is the function that specifies the values of initial tokens.

There is at least one self-loop per SDFA, i.e. for all $a \in I^{\mathrm{A}}$ there exists $b \in I^{\mathrm{B}}$, such that

$$(a, N_{\mathscr{Y}_a}) = \mathrm{src}(b), \quad (a, N_{\mathscr{U}_a}) = \mathrm{dst}(b), \quad \mathrm{d}_0(b) = 1 \tag{3.8}$$

An SDFG is weakly connected, i.e. there exists an undirected path between any two nodes of the graph.

If an SDFG is not connected, it can be modeled as two separate SDFGs. An SDFG is defined as connected because condition forms the basis for the analysis shown in [35]. This

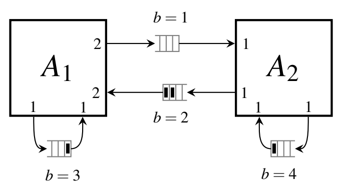


Figure 3.1: The SDFG presented in Example 3.3.

analysis finds sufficient and necessary conditions for a non-terminating execution of an SDFG. The execution of an SDFG will be covered in the next two sections and in the next chapter.

Example 3.3 (SDF graph)**.** This example specifies an SDFG (Definition 3.2) shown in Figure 3.1. The set of SDFA labels $I^{\mathrm{A}} \subset \mathbb{N}$ corresponds to

$$I^{\mathrm{A}} = \{1,2\} \tag{3.9}$$

The function for labeling SDFAs $\mathrm{A} : I^{\mathrm{A}} \to \mathcal{A}$ is defined by

$$\mathrm{A}(a) = \begin{cases} A_1, & a = 1 \\ A_2, & a = 2 \end{cases} \tag{3.10}$$

where

$$A_1 = \left(I_{\mathscr{U}_1}, I_{\mathscr{Y}_1}, \mathscr{U}_1, \mathscr{Y}_1, \mathrm{c}_1, \mathrm{p}_1, \mathrm{calculate}_1\right) \tag{3.11a}$$

$$I_{\mathscr{U}_1} = \{1,2\}, \quad I_{\mathscr{Y}_1} = \{1,2\} \tag{3.11b}$$

$$\mathscr{U}_1(\iota) = \mathbb{Q}, \quad \iota = 1,2 \tag{3.11c}$$

$$\mathscr{Y}_1(l) = \mathbb{Q}, \quad l = 1,2 \tag{3.11d}$$

$$\mathrm{c}_1(\iota) = \begin{cases} 2, & \iota = 1 \\ 1, & \iota = 2 \end{cases}, \quad \mathrm{p}_1(l) = \begin{cases} 2, & l = 1 \\ 1, & l = 2 \end{cases} \tag{3.11e}$$

and

$$A_2 = \left(I_{\mathscr{U}_2}, I_{\mathscr{Y}_2}, \mathscr{U}_2, \mathscr{Y}_2, \mathrm{c}_2, \mathrm{p}_2, \mathrm{calculate}_2\right) \tag{3.12a}$$

$$I_{\mathscr{U}_2} = \{1,2\}, \quad I_{\mathscr{Y}_2} = \{1,2\} \tag{3.12b}$$

$$\mathscr{U}_2(\iota) = \mathbb{R}, \quad \iota = 1,2 \tag{3.12c}$$

$$\mathscr{Y}_2(l) = \mathbb{R}, \quad l = 1,2 \tag{3.12d}$$

$$\mathrm{c}_2(\iota) = \begin{cases} 1, & \iota = 1 \\ 1, & \iota = 2 \end{cases}, \quad \mathrm{p}_2(l) = \begin{cases} 1, & l = 1 \\ 1, & l = 2 \end{cases} \tag{3.12e}$$

$$\tag{3.12f}$$

The calculation function of the SDFA A_1

$$\mathrm{calculate}_1 : \mathcal{Z}^{I_{\mathscr{U}_1} \times \mathbb{N}} \to \mathcal{Z}^{I_{\mathscr{Y}_1} \times \mathbb{N}} \tag{3.13}$$

consumes three tokens and produces three tokens

$$\gamma_1(k_1) = \mathrm{calculate}_1\left(\xi_1(k_1)\right) \tag{3.14}$$

where

$$\begin{aligned} \gamma_1(k_1,1,1) &= \frac{1}{2}\,\xi_1(k_1,1,1) + \frac{1}{3}\,\xi_1(k_1,1,2) + \frac{1}{5}\,\xi_1(k_1,2,1) \\ \gamma_1(k_1,1,2) &= \frac{1}{7}\,\xi_1(k_1,1,1) + \frac{1}{11}\,\xi_1(k_1,1,2) + \frac{1}{13}\,\xi_1(k_1,2,1) \\ \gamma_1(k_1,2,1) &= \frac{1}{17}\,\xi_1(k_1,1,1) + \frac{1}{19}\,\xi_1(k_1,1,2) + \frac{1}{23}\,\xi_1(k_1,2,1) \end{aligned} \tag{3.15}$$

The calculation function of the SDFA A_2

$$\mathrm{calculate}_2 : \mathcal{Z}^{I_{\mathscr{U}_2} \times \mathbb{N}} \to \mathcal{Z}^{I_{\mathscr{Y}_2} \times \mathbb{N}} \tag{3.16}$$

consumes two tokens and produces two tokens

$$\gamma_2(k_2) = \mathrm{calculate}_2\left(\xi_2(k_2)\right) \tag{3.17}$$

where

$$\begin{aligned} \gamma_2(k_2,1,1) &= \frac{1}{29}\,\xi_2(k_2,1,1) + \frac{1}{31}\,\xi_2(k_2,2,1) \\ \gamma_2(k_2,2,1) &= \frac{1}{37}\,\xi_2(k_2,1,1) + \frac{1}{41}\,\xi_2(k_2,2,1) \end{aligned} \tag{3.18}$$

The set of SDFB labels $I^{\mathrm{B}} \subset \mathbb{N}$ corresponds to

$$I^{\mathrm{B}} = \{1,2,3,4\} \tag{3.19}$$

The function $\mathrm{src} : I^{\mathrm{B}} \to I^{\mathrm{A}}$ is given by

$$\mathrm{src}(b) = \begin{cases} (1,1), & b=1 \\ (2,1), & b=2 \\ (1,2), & b=3 \\ (2,2), & b=4 \end{cases} \tag{3.20}$$

and the function $\mathrm{dst} : I^{\mathrm{B}} \to I^{\mathrm{A}}$ is given by

$$\mathrm{dst}(b) = \begin{cases} (2,1), & b=1 \\ (1,1), & b=2 \\ (1,2), & b=3 \\ (2,2), & b=4 \end{cases} \tag{3.21}$$

The number of initial tokens $\mathrm{d}_0 : I^{\mathrm{B}} \to \mathbb{N}$ is given by

$$\mathrm{d}_0(b) = \begin{cases} 0, & b=1 \\ 2, & b=2 \\ 1, & b=3 \\ 1, & b=4 \end{cases} \tag{3.22}$$

The values initial token values $\omega_0 : I^{\mathrm{B}} \to \mathcal{Z}^{\mathbb{N}}$ are given by

$$\omega_0(b,\theta) = \begin{cases} \frac{1}{43}, & b=2, \quad \theta=1 \\ \frac{1}{47}, & b=2, \quad \theta=2 \\ \frac{1}{53}, & b=3, \quad \theta=1 \\ \frac{1}{59}, & b=4, \quad \theta=1 \end{cases} \tag{3.23}$$

Definitions 3.1 and 3.2 define how to construct an SDFG. Definitions 3.4-3.6 define the terms used in the description of the SDFG execution (Definition 3.7). The next definition introduces the sequence of operations during SDFG execution.

Definition 3.4 (Sequence of operations)**.** The function that consumes the tokens from the SDFBs and prepares the tokens for calculation by an SDFA is

$$\begin{aligned}\text{consume} : &\left(I^{\mathrm{B}} \to \mathbb{N}_0\right) \times \left(I^{\mathrm{B}} \to \mathcal{Z}^{\mathbb{N}}\right) \times I^{\mathrm{A}} \\ &\to \left(I^{\mathrm{B}} \to \mathbb{N}_0\right) \times \left(I^{\mathrm{B}} \to \mathcal{Z}^{\mathbb{N}}\right) \times \mathcal{Z}^{\mathbb{N}\times\mathbb{N}}\end{aligned} \tag{3.24}$$

The function that transfers the tokens produced by an SDFA to the SDFBs is

$$\begin{aligned}\text{produce} : &\left(I^{\mathrm{B}} \to \mathbb{N}_0\right) \times \left(I^{\mathrm{B}} \to \mathcal{Z}^{\mathbb{N}}\right) \times I^{\mathrm{A}} \times \mathcal{Z}^{\mathbb{N}\times\mathbb{N}} \\ &\to \left(I^{\mathrm{B}} \to \mathbb{N}_0\right) \times \left(I^{\mathrm{B}} \to \mathcal{Z}^{\mathbb{N}}\right)\end{aligned} \tag{3.25}$$

The sequence of operations used when executing an SDFG is given by the function

$$\text{operation} : \mathbb{N} \to \{\text{consume}, \text{produce}\} \times I^{\mathrm{A}} \tag{3.26}$$

Definition 3.5 (Token state)**.** The number of tokens in SDFBs during the execution of an SDFG is indicated by

$$\mathrm{d} : \mathbb{N}_0 \to {\mathbb{N}_0}^{I^{\mathrm{B}}} \tag{3.27}$$

The token values in SDFBs during the execution of an SDFG are indicated by

$$\omega : \mathbb{N}_0 \to \left(I^{\mathrm{B}} \to \mathcal{Z}^{\mathbb{N}}\right) \tag{3.28}$$

The value of a token in an SDFB after the n^{th} SDFG execution step can be obtained by $\omega(n, b, \theta) \in \mathcal{Z}$ where $b \in I^{\mathrm{B}}$ and $\theta \leqslant \mathrm{d}(n)$.

Definition 3.6 (Number of operations)**.** The number of SDFA operations after an SDFG execution step is given by

$$\text{num} : \{\text{consume}, \text{produce}\} \times I^{\mathrm{A}} \times \mathbb{N}_0 \to \mathbb{N} \tag{3.29}$$

where

$$\text{num}(op, a, n) = \begin{cases} 0, & n = 0 \\ \text{num}(op, a, n-1) + \text{count}(op, a, n), & n > 0 \end{cases} \tag{3.30}$$

and

$$\text{count}(op, a, n) = \begin{cases} 1, & \text{operation}(n) = (op, a) \\ 0, & otherwise \end{cases} \tag{3.31}$$

Definition 3.7 (SDFG execution)**.** The number of tokens is initially set to

$$\mathrm{d}(0) = \mathrm{d}_0 \tag{3.32}$$

and the token values to

$$\omega(0) = \omega_0 \tag{3.33}$$

If

$$\text{operation}(n) = (\text{consume}, a), \quad a \in I^{\mathrm{A}} \tag{3.34}$$

then for all $\iota \in I_{\mathscr{U}_a}$

$$\mathrm{d}(n-1, b) \geqslant \mathrm{c}_a(\iota), \quad (a, \iota) = \mathrm{dst}(b) \tag{3.35}$$

For all $a \in I^{\mathrm{A}}$ and all $n \in \mathbb{N}$ the number of token consumptions is greater than or equal the number of token productions

$$\text{num}(\text{consume}, a, n) \geqslant \text{num}(\text{produce}, a, n) \tag{3.36}$$

If the condition (3.35) holds for $a \in I^{\mathrm{A}}$ and $n \in \mathbb{N}$, there exist $n', n'' \in \mathbb{N}$ for which

$$n'' > n' > n \tag{3.37}$$

and

$$\begin{aligned} (\text{consume}, a) &= \text{operation}\left(n'\right) \\ (\text{produce}, a) &= \text{operation}\left(n''\right) \end{aligned} \tag{3.38}$$

When the SDFA is triggered, tokens from the SDFBs are consumed

$$\Big(\mathrm{d}(n'), \omega(n'), \xi_a\big(\text{num}(\text{consume}, a, n')\big)\Big) = \text{consume}\big(\mathrm{d}(n'-1), \omega(n'-1), a\big) \tag{3.39}$$

where consumed tokens are

$$\xi_a\big(\text{num}(\text{consume}, a, n'), \iota, \theta\big) = \omega\left(n'-1, b, \theta\right), \quad (a, \iota) = \mathrm{dst}(b), \quad \theta \leqslant \mathrm{c}(\iota) \tag{3.40}$$

After the production of the tokens

$$\gamma_a\big(\text{num}(\text{produce}, a, n'')\big) = \text{calculate}_a\Big(\xi_a\big(\text{num}(\text{consume}, a, n'')\big)\Big) \tag{3.41}$$

the produced tokens are transferred to the SDFBs

$$\big(\mathrm{d}(n''), \omega(n'')\big) = \text{produce}\Big(\mathrm{d}(n''-1), \omega(n''-1), a, \gamma_a\big(\text{num}(\text{produce}, a, n'')\big)\Big) \tag{3.42}$$

The number of tokens in the SDFBs during the SDFG execution is

$$\mathrm{d}(n,b) = \begin{cases} \mathrm{d}(n-1,b) - \mathrm{c}_a(\iota), & \begin{array}{l}\mathrm{operation}(n) = (\mathrm{consume}, a) \\ (a,\iota) = \mathrm{dst}(b)\end{array} \\ \hline \mathrm{d}(n-1,b) + \mathrm{p}_a(o), & \begin{array}{l}\mathrm{operation}(n) = (\mathrm{produce}, a) \\ (a,o) = \mathrm{src}(b)\end{array} \\ \hline \mathrm{d}(n-1,b), & \mathit{otherwise} \end{cases} \tag{3.43}$$

Values of the tokens in the SDFBs are

$$\omega(n,b,\theta) = \begin{cases} \omega\big(n-1,b,\theta+\mathrm{c}_a(\iota)\big), & \begin{array}{l}\mathrm{operation}(n) = (\mathrm{consume}, a) \\ (a,\iota) = \mathrm{dst}(b) \\ \theta \leqslant \mathrm{d}(n,b)\end{array} \\ \hline \gamma_a\big(\mathrm{num}(\mathrm{produce},a,n),o,\theta-\mathrm{d}(n-1,b)\big), & \begin{array}{l}\mathrm{operation}(n) = (\mathrm{produce}, a) \\ (a,o) = \mathrm{src}(b), \\ \mathrm{d}(n-1,b) < \theta \leqslant \mathrm{d}(n,b)\end{array} \\ \hline \omega(n-1,b,\theta), & \mathit{otherwise} \end{cases} \tag{3.44}$$

The above definition introduces rules for the execution of an SDFG (Definition 3.2). An algorithm that executes the SDFG can perform token consumptions, SDFA calculations, and token productions in any order that conforms to the above rules. The topic of the next section is to prove that every algorithm that correctly executes the SDFG produces the same results. The execution algorithm can trigger an SDFA as soon as all of the SDFBs connected to its input ports have enough tokens (3.35). When the SDFA is triggered, the tokens are consumed (3.39), new tokens are produced with the SDFA calculation (3.41), and produced tokens are transferred to the SDFBs (3.42). The token consumption forms input tokens for the calculation (3.40), reduces the number of tokens in the SDFBs (3.43) and modifies the state in the SDFBs (3.44). The transfer of produced tokens increases the number of tokens in the SDFBs (3.43) and modifies the state in the SDFBs (3.44).

The equations (3.43) and (3.44) define the behavior of the SDFBs. An SDFB is a first-in first-out queue. The tokens that come into the queue should leave the queue in the same order. SDFBs are an important building block of an SDFG as they maintain the order of tokens exchanged between SDFAs.

3.2 Determinacy

An SDFG is a determinate MOC [34]. Any algorithm that executes it correctly (Definition 3.7) should produce the same results. Theorem 3.10 and Theorem 3.14 demonstrate this statement. The analysis presented in this section closely follows [34]. It is reformulated because SDFGs (Definition 3.2) in this thesis are defined so that they have at least one self-loop (3.8) and the execution model (Definition 3.4) is different.

The condition (3.35) and the self-loop (3.8) ensure that two invocations of the same SDFA cannot be executed at the same time. The authors in [28] suggest that self-loops are a way to ensure the integrity of the SDFBs and are required if an SDFA has a state. As will be seen in Section 3.4, all simulators are stateful. The self-loop assumption is used in the proof of Lemma 3.13. This lemma enables the calculation to be rewritten as (3.91), which is used to prove determinate execution (Theorem 3.14).

Lemma 3.8 (Number of tokens in the SDFBs)**.** *Assume that an SDFG (Definition 3.2) is executed with a valid execution (Definition 3.7). For all $b \in I^{\mathrm{B}}$*

$$\mathrm{d}(n,b) = \mathrm{d}_0(b) + \mathrm{num}(\mathrm{produce},\hat{a},n)\ \mathrm{p}_{\hat{a}}(\hat{o}) - \mathrm{num}(\mathrm{consume},\check{a},n)\ \mathrm{c}_{\check{a}}(\check{\imath}) \tag{3.45}$$

where $\hat{a},\check{a} \in I^{\mathrm{A}}$ and

$$(\hat{a},\hat{o}) = \mathrm{src}(b), \quad (\check{a},\check{\imath}) = \mathrm{dst}(b), \quad \hat{o} \in I_{\mathscr{Y}_{\hat{a}}}, \quad \check{\imath} \in I_{\mathscr{U}_{\check{a}}} \tag{3.46}$$

Proof. The proof is given by induction. The induction basis is the case of $n = 0$. It is proved by (3.30) and (3.32).

In the induction step it is assumed that (3.45) holds. The induction step is proved by (3.30), (3.43), and (3.45). If operation$(n) = (\mathrm{consume},\check{a})$ then

$$\begin{aligned}\mathrm{d}(n+1,b) &= \mathrm{d}(n,b) - \mathrm{c}_{\check{a}}(\check{\imath}) \\ &= \mathrm{d}_0(b) + \mathrm{num}(\mathrm{produce},\hat{a},n)\ \mathrm{p}_{\hat{a}}(\hat{o}) - \mathrm{num}(\mathrm{consume},\check{a},n)\ \mathrm{c}_{\check{a}}(\check{\imath}) - \mathrm{c}_{\check{a}}(\check{\imath}) \\ &= \mathrm{d}_0(b) + \mathrm{num}(\mathrm{produce},\hat{a},n+1)\ \mathrm{p}_{\hat{a}}(\hat{o}) - \mathrm{num}(\mathrm{consume},\check{a},n+1)\ \mathrm{c}_{\check{a}}(\check{\imath})\end{aligned} \tag{3.47}$$

If operation $(n) = (\text{produce}, \hat{a})$ then

$$\begin{aligned} \mathrm{d}(n+1,b) &= \mathrm{d}(n,b) + \mathrm{p}_{\hat{a}}(\hat{o}) \\ &= \mathrm{d}_0(b) + \mathrm{num}(\text{produce}, \hat{a}, n)\ \mathrm{p}_{\hat{a}}(\hat{o}) - \mathrm{num}(\text{consume}, \check{a}, n)\ \mathrm{c}_{\check{a}}(\check{\iota}) + \mathrm{p}_{\hat{a}}(\hat{o}) \\ &= \mathrm{d}_0(b) + \mathrm{num}(\text{produce}, \hat{a}, n+1)\ \mathrm{p}_{\hat{a}}(\hat{o}) - \mathrm{num}(\text{consume}, \check{a}, n+1)\ \mathrm{c}_{\check{a}}(\check{\iota}) \end{aligned} \tag{3.48}$$

If

$$\text{operation}(n) = (op, a'), \quad a' \neq \hat{a}, \quad a' \neq \check{a}, \quad op \in \{\text{consume}, \text{produce}\}, \quad a' \in I^{\mathrm{A}} \tag{3.49}$$

then

$$\begin{aligned} \mathrm{d}(n+1,b) &= \mathrm{d}(n,b) \\ &= \mathrm{d}_0(b) + \mathrm{num}(\text{produce}, \hat{a}, n)\ \mathrm{p}_{\hat{a}}(\hat{o}) - \mathrm{num}(\text{consume}, \check{a}, n)\ \mathrm{c}_{\check{a}}(\check{\iota}) \\ &= \mathrm{d}_0(b) + \mathrm{num}(\text{produce}, \hat{a}, n+1)\ \mathrm{p}_{\hat{a}}(\hat{o}) - \mathrm{num}(\text{consume}, \check{a}, n+1)\ \mathrm{c}_{\check{a}}(\check{\iota}) \end{aligned} \tag{3.50}$$

□

Lemma 3.9 (Trigger condition)**.** *Assume that an SDFG (Definition 3.2) is correctly executed (Definition 3.7). If*

$$\text{operation}(n) = (\text{consume}, a), \quad a \in I^{\mathrm{A}} \tag{3.51}$$

then for all $\iota \in I_{\mathscr{U}_a}$

$$\mathrm{num}(\text{consume}, a, n-1)\ \mathrm{c}_a(\iota) + \mathrm{c}_a(\iota) \leqslant \mathrm{d}_0(b) + \mathrm{num}(\text{produce}, \hat{a}, n-1)\ \mathrm{p}_{\hat{a}}(\hat{o}) \tag{3.52}$$

where

$$(\hat{a}, \hat{o}) = \mathrm{src}(b), \quad (a, \iota) = \mathrm{dst}(b) \tag{3.53}$$

Proof. The condition (3.52) follows from Lemma 3.8 and (3.35). □

Theorem 3.10 (Number of executions)**.** *Assume that an SDFG (Definition 3.2) can be executed (Definition 3.7) with two different sequences of operations* operation$'$ *and* operation$''$ (3.26). *Let* num$'$ *and* num$''$ *be the number of operations performed* (3.30) *for the respective execution. For all $a \in I^{\mathrm{A}}$, all operations $op \in \{\text{consume}, \text{produce}\}$, and all $n' \in \mathbb{N}$, there exists an $n'' \in \mathbb{N}$ such that*

$$\mathrm{num}'(op, a, n') = \mathrm{num}''(op, a, n'') \tag{3.54}$$

Proof. The proof is given by contradiction. Let $n'_0 \in \mathbb{N}$ denote the least number such that

$$\mathrm{num}'(op_0, a_0, n'_0) > \mathrm{num}''(op_0, a_0, n'') \tag{3.55}$$

for some $op_0 \in \{\text{consume}, \text{produce}\}$, for some $a_0 \in I^{\mathrm{A}}$, and for all $n'' \in \mathbb{N}$. From (3.36) it follows that

$$\text{operation}'(n_0) = (op_0, a_0) = (\text{consume}, a_0) \tag{3.56}$$

Consequently

$$\text{num}'\left(\text{consume}, a_0, n_0'\right) = \text{num}'\left(\text{consume}, a_0, n_0' - 1\right) + 1 \tag{3.57}$$

follows from (3.30). The trigger condition (Lemma 3.9) states that for all $\iota_0 \in I_{\mathscr{U}_{a_0}}$

$$\text{num}'\left(\text{consume}, a_0, n_0' - 1\right)\ \mathrm{c}_{a_0}(\iota_0) + \mathrm{c}_{a_0}(\iota_0) \leqslant \mathrm{d}_0(b_0) + \text{num}'\left(\text{produce}, \hat{a}_0, n_0' - 1\right)\ \mathrm{p}_{\hat{a}_0}(\hat{o}_0) \tag{3.58}$$

where

$$(\hat{a}_0, \hat{o}_0) = \text{src}(b_0), \quad (a_0, \iota_0) = \text{dst}(b_0) \tag{3.59}$$

Since n_0' is minimal, the contradiction assumption (3.55) does not apply to $n_0' - 1$, i.e. for some $n_0'' \in \mathbb{N}$, for all $\iota_0 \in I_{\mathscr{U}_{a_0}}$

$$\text{num}'\left(\text{consume}, a_0, n_0' - 1\right) \leqslant \text{num}''\left(\text{consume}, a_0, n_0''\right) \tag{3.60}$$

and

$$\text{num}'\left(\text{produce}, \hat{a}_0, n_0' - 1\right) \leqslant \text{num}''\left(\text{produce}, \hat{a}_0, n_0''\right) \tag{3.61}$$

where

$$(\hat{a}_0, \hat{o}_0) = \text{src}(b_0), \quad (a_0, \iota_0) = \text{dst}(b_0) \tag{3.62}$$

From (3.55), (3.57) and (3.60) it follows that

$$\text{num}'\left(\text{consume}, a_0, n_0' - 1\right) = \text{num}''\left(\text{consume}, a_0, n_0''\right) \tag{3.63}$$

From (3.58), (3.61) and (3.63) it follows that the trigger condition

$$\text{num}''\left(\text{consume}, a_0, n_0''\right)\ \mathrm{c}_{a_0}(\iota_0) + \mathrm{c}_{a_0}(\iota_0) \leqslant \mathrm{d}_0(b) + \text{num}''\left(\text{produce}, \hat{a}_0, n_0''\right)\ \mathrm{p}_{\hat{a}_0}(\hat{o}_0) \tag{3.64}$$

is met for all $\iota_0 \in I_{\mathscr{U}_{a_0}}$ where

$$(\hat{a}_0, \hat{o}_0) = \text{src}(b_0), \quad (a_0, \iota_0) = \text{dst}(b_0) \tag{3.65}$$

By condition (3.38) there exists $n''_{next} > n_0''$ such that

$$\text{operation}\left(n''_{next}\right) = (\text{consume}, a_0) \tag{3.66}$$

From (3.57) and (3.63) it follows that

$$\mathrm{num}''\left(\text{consume}, a_0, n''_{next}\right) \geqslant \mathrm{num}''\left(\text{consume}, \hat{a}_0, n''_0\right) + 1 = \mathrm{num}'\left(\text{consume}, \hat{a}_0, n'_0\right) \tag{3.67}$$

which contradicts (3.55) and completes the proof. □

Lemma 3.11 (Visiting tokens). *Let $\omega_{\text{total}} : I^{\text{B}} \times \mathbb{N}_0 \to \mathbb{N}_0$ be the sequence of tokens visiting the SDFBs*

$$\omega_{\text{total}}(b, \theta_{total}) = \begin{cases} \omega_0\left(b, \theta_{total}\right), & \theta_{total} \leqslant \mathrm{d}_0(b) \\ \hline \gamma_{\hat{a}}\left(k_{\hat{a}}, \hat{o}, \theta_{\hat{a}}\right), & \begin{aligned} & k_{\hat{a}} = \left\lceil \frac{\theta_{total} - \mathrm{d}_0(b)}{\mathrm{p}_{\hat{a}}(\hat{o})} \right\rceil \\ & \theta_{\hat{a}} = \theta_{total} - \mathrm{d}_0(b) - (k_{\hat{a}} - 1)\ \mathrm{p}_{\hat{a}}(\hat{o}) \\ & (\hat{a}, \hat{o}) = \mathrm{src}(b) \end{aligned} \end{cases} \tag{3.68}$$

For all $b \in I^{\text{B}}$

$$\omega\left(n, b, \theta\right) = \omega_{\text{total}}\big(b, \theta + k_{\breve{a}}\ \mathrm{c}_{\breve{a}}(\breve{\imath})\big), \quad \begin{aligned} & k_{\breve{a}} = \mathrm{num}\left(\text{consume}, \breve{a}, n\right) \\ & (\breve{a}, \breve{\imath}) = \mathrm{dst}(b) \end{aligned} \tag{3.69}$$

Proof. The induction basis is the case of $n = 0$. It is proven by (3.33) and (3.68).

In the induction step it is assumed that (3.69) holds for $n \in \mathbb{N}_0$. Let $b \in I^{\text{B}}$ be fixed and

$$\begin{aligned} (\hat{a}, \hat{o}) &= \mathrm{src}(b) \\ (\breve{a}, \breve{\imath}) &= \mathrm{dst}(b) \end{aligned} \tag{3.70}$$

In case of

$$\text{operation}\left(n+1\right) = \left(\text{consume}, \breve{a}\right) \tag{3.71}$$

the statement

$$\begin{aligned} \omega\left(n+1, b, \theta\right) &= \omega\big(n, b, \theta + \mathrm{c}_{\breve{a}}(\breve{\imath})\big) \\ &= \omega_{\text{total}}\big(b, \theta + \left(k_{\breve{a}} + 1\right)\ \mathrm{c}_{\breve{a}}(\breve{\imath})\big) \end{aligned} \tag{3.72}$$

follows from (3.44), (3.69) and

$$\begin{aligned} k_{\breve{a}} &= \mathrm{num}\left(\text{consume}, \breve{a}, n\right) \\ k_{\breve{a}} + 1 &= \mathrm{num}\left(\text{consume}, \breve{a}, n+1\right) \end{aligned} \tag{3.73}$$

In case of

$$\text{operation}\left(n+1\right) = \left(\text{produce}, \hat{a}\right) \tag{3.74}$$

the statement

$$
\begin{aligned}
\omega\left(n+1,b,\theta\right) &= \begin{cases} \gamma_{\hat{a}}\big(k_{\hat{a}}+1,\hat{o},\theta-\mathrm{d}(n,b)\big), & \mathrm{d}(n,b)<\theta\leqslant\mathrm{d}(n+1,b)\\ \omega(n,b,\theta), & \theta\leqslant\mathrm{d}(n,b)\end{cases}\\
&= \begin{cases} \gamma_{\hat{a}}\big(k_{\hat{a}}+1,\hat{o},\theta_{\hat{a}}\big), & \theta_{\hat{a}}=\theta-\mathrm{d}_0(b)-k_{\hat{a}}\,\mathrm{p}_{\hat{a}}(\hat{o}),\quad 1\leqslant\theta_{\hat{a}}\leqslant\mathrm{p}_{\hat{a}}(\hat{o})\\ \omega(n,b,\theta), & \theta\leqslant\mathrm{d}(n,b)\end{cases}\\
&= \omega_{\text{total}}\big(b,\theta+k_{\breve{a}}\,\mathrm{c}_{\breve{a}}(\breve{\imath})\big)
\end{aligned}
\tag{3.75}
$$

follows from (3.44), (3.45), (3.68), (3.69) and

$$
\begin{aligned}
&k_{\breve{a}}=\operatorname{num}\left(\text{consume},\breve{a},n+1\right)=\operatorname{num}\left(\text{consume},\breve{a},n\right)\\
&k_{\hat{a}}=\operatorname{num}\left(\text{produce},\hat{a},n\right)\\
&k_{\hat{a}}+1=\left\lceil\frac{\theta+k_{\breve{a}}\,\mathrm{c}_{\breve{a}}(\breve{\imath})-\mathrm{d}_0(b)}{\mathrm{p}_{\hat{a}}(\hat{o})}\right\rceil=\operatorname{num}\left(\text{produce},\hat{a},n+1\right),\\
&\mathrm{d}(n,b)<\theta\leqslant\mathrm{d}(n+1,b)
\end{aligned}
\tag{3.76}
$$

In case of

$$
\operatorname{operation}\left(n+1\right)=\left(op,a\right),\quad op\in\left(\text{consume},\text{produce}\right),\quad a\neq\hat{a},\quad a\neq\breve{a}
\tag{3.77}
$$

the statement

$$
\omega(n+1,b,\theta)=\omega_{\text{total}}(b,\theta+k_{\breve{a}}\,\mathrm{c}_{\breve{a}}(\breve{\imath}))
\tag{3.78}
$$

follows from (3.44), (3.69) and

$$
k_{\breve{a}}=\operatorname{num}\left(\text{consume},\breve{a},n+1\right)=\operatorname{num}\left(\text{consume},\breve{a},n\right)
\tag{3.79}
$$

□

Lemma 3.12 (Input tokens)**.** *Assume that an SDFG (Definition 3.2) is executed with a valid execution (Definition 3.7). Let*

$$
\operatorname{operation}(n)=(\text{consume},a),\quad k_{\breve{a}}=\operatorname{num}(\text{consume},a,n)
\tag{3.80}
$$

The input token values are

$$\xi_{\breve{a}}\left(k_{\breve{a}}, \breve{\imath}, \theta_{\breve{a}}\right) = \begin{cases} \omega_0\big(b, \theta_{\breve{a}} + (k_{\breve{a}} - 1)\, \mathrm{c}_{\breve{a}}(\breve{\imath})\big), & \theta_{\breve{a}} + (k_{\breve{a}} - 1)\, \mathrm{c}_{\breve{a}}(\breve{\imath}) \leqslant \mathrm{d}_0(b) \\ \gamma_{\hat{a}}(k_{\hat{a}}, \hat{o}, \theta_{\hat{a}}), & \begin{array}{l} \theta_{\breve{a}} + (k_{\breve{a}} - 1)\, \mathrm{c}_{\breve{a}}(\breve{\imath}) > \mathrm{d}_0(b), \\ k_{\hat{a}} = \left\lceil \frac{\theta_{\breve{a}} + (k_{\breve{a}} - 1)\, \mathrm{c}_{\breve{a}}(\breve{\imath}) - \mathrm{d}_0(b)}{\mathrm{p}_{\hat{a}}(\hat{o})} \right\rceil, \\ \theta_{\hat{a}} = \theta_{\breve{a}} + (k_{\breve{a}} - 1)\, \mathrm{c}_{\breve{a}}(\breve{\imath}) - \mathrm{d}_0(b) - (k_{\hat{a}} - 1)\, \mathrm{p}_{\hat{a}}(\hat{o}) \end{array} \end{cases} \tag{3.81}$$

where

$$(\hat{a}, \hat{o}) = \mathrm{src}(b), \quad (\breve{a}, \breve{\imath}) = \mathrm{dst}(b) \tag{3.82}$$

Proof. The proof uses Lemma 3.11. From (3.40) and (3.69) it follows that

$$\xi_{\breve{a}}(k_{\breve{a}}, \breve{\imath}, \theta_{\breve{a}}) = \omega(n-1, b, \theta_{\breve{a}}) = \omega_{\mathrm{total}}\big(b, \theta_{\breve{a}} + (k_{\breve{a}} - 1)\, \mathrm{c}_{\breve{a}}(\breve{\imath})\big) \tag{3.83}$$

The expression (3.81) follows from (3.68). □

Lemma 3.13 (Consistent token production)**.** *Assume that an SDFG (Definition 3.2) is executed with a valid execution (Definition 3.7). For all $n \in \mathbb{N}$ if*

$$\mathrm{operation}(n) = (\mathrm{produce}, a) \tag{3.84}$$

then

$$\mathrm{num}(\mathrm{produce}, a, n) = \mathrm{num}(\mathrm{consume}, a, n) \tag{3.85}$$

Proof. Let

$$\mathrm{operation}(n_c) = (\mathrm{consume}, a) \tag{3.86}$$

where

$$\mathrm{num}(\mathrm{consume}, a, n_c) = \mathrm{num}(\mathrm{consume}, a, n) \tag{3.87}$$

From Lemma 3.9 and (3.8) it follows that

$$\mathrm{num}(\mathrm{consume}, a, n) \leqslant 1 + \mathrm{num}(\mathrm{produce}, a, n_c) \tag{3.88}$$

From (3.30) and (3.84) it follows that

$$\mathrm{num}(\mathrm{produce}, a, n_c) < \mathrm{num}(\mathrm{produce}, a, n) \tag{3.89}$$

and consequently

$$\mathrm{num}(\mathrm{consume}, a, n) < 1 + \mathrm{num}(\mathrm{produce}, a, n) \tag{3.90}$$

The equation (3.85) follows from (3.36) and (3.90). □

Lemma 3.13 enables rewriting (3.41) as

$$\gamma_a(k_a) = \text{calculate}_a\big(\xi_a(k_a)\big) \tag{3.91}$$

For this reason, the lemma is called "Consistent token production". Any valid SDFG execution (Definition 3.7) calculates the output tokens from the input tokens with the same index. It shows the importance of self-loops (3.8). Without them, it would be possible to construct an example where two executions of the same SDFG give different results.

Theorem 3.14 (Determinate execution)**.** *Assume that an SDFG (Definition 3.2) can be executed (Definition 3.7) with two different sequences of operations* $\text{operation}'$ *and* $\text{operation}''$ (3.26). *Let* ξ'_a *and* ξ''_a *denote the sequences of input tokens* (3.40) *for the respective execution. Let* γ'_a *and* γ''_a *denote the sequences of output tokens* (3.40) *for the respective execution. For all* $a \in I^{\mathrm{A}}$*, for all* $\iota \in I_{\mathscr{U}_a}$ *and for all valid* $k_a \in \mathbb{N}$

$$\xi'_a(k_a, \iota, \theta) = \xi''_a(k_a, \iota, \theta), \quad 1 \leqslant \theta \leqslant \mathrm{c}_a(\iota) \tag{3.92}$$

$$\gamma'_a(k_a, o, \theta) = \gamma''_a(k_a, o, \theta), \quad 1 \leqslant \theta \leqslant \mathrm{p}_a(\iota) \tag{3.93}$$

Proof. The proof is given by contradiction. Let $k_{a0} \in \mathbb{N}$ denote the least number for which one of (3.92) or (3.93) does not hold. If (3.92) does not hold, this contradicts Lemma 3.12, since input tokens are completely determined by the previously generated tokens. If (3.93) does not hold, this contradicts Lemma 3.13, since output tokens are determined by the input tokens (3.91). □

This section was included in the work to highlight the importance of determinacy. Determinacy is a very useful property of a MOC [22, 23, 24]. If the behavior of the model is not fully specified in practice, the results will vary depending on which engineering team solves the problem. Examples of problems that can arise are racing conditions [25, 26]. The analysis of the SDFG execution can be carried out with a sequential algorithm due to the determinacy. If the sequential algorithm deadlocks, the parallel one also deadlocks [35].

Since SDFG is determinate, a co-simulation engineer should be able to reproduce the results of any experiment using the SDFG. This allows engineers to exchange SDFGs instead of large amounts of data. In addition, the results do not need to be reproduced on the same platform. A program that runs an SDFG on a single processor is likely to produce results more slowly than a program that runs on multiple processors. The results will be identical. This makes SDFG a good candidate for specifying co-simulation experiments.

3.3 Periodic admissible sequential schedule

This section introduces periodic adimissible sequential schedule (Definition 3.15) for the sequential execution (Algorithm 3.1) of an SDFG (Definition 3.2). Since SDFG is a determinate MOC [34], sequential execution produces the same results as any other execution. This makes the sequential execution analysis an important tool for checking whether any valid execution of an SDFG has a deadlock (Example 3.19). Necessary and sufficient conditions for the existence of periodic adimissible sequential schedule (PASS) are analyzed in [35]. Algorithm 3.2 can be used to check whether a PASS exists, i.e. whether sequential execution deadlocks. If sequential execution deadlocks, any other execution should also deadlock.

Algorithm 3.1 Sequential execution of an SDF graph (SDFG)

```
Require: $G \in \mathcal{G}$ (Definition 3.2),   $\sigma$ is a PASS
  $\omega(0) := \omega_0, \quad \mathrm{d}(0) := \mathrm{d}_0, \quad n := 0$
  for $a \in I^{\mathrm{A}}$ do
      $k_a := 0$
  repeat
      $n := n+1, \quad a := \sigma(n), \quad k_a := k_a + 1$
      $\big(\mathrm{d}(2n+1), \omega(2n+1), \xi_a(k_a)\big) := \mathrm{consume}\big(\mathrm{d}(2n), \omega(2n), a\big)$
      $\gamma_a(k_a) := \mathrm{calculate}_a\big(\xi_a(k_a)\big)$
      $\big(\mathrm{d}(2n+2), \omega(2n+2)\big) := \mathrm{produce}\big(\mathrm{d}(2n+1), \omega(2n+1), a, \gamma_a(k_a)\big)$
  until external stop
```

Definition 3.15 (Periodic adimissible sequential schedule)**.** A periodic adimissible sequential schedule (PASS) is a sequence $\sigma : \mathbb{N} \to I^{\mathrm{A}}$ which gives the index of an SDFA executed in the n^{th} step of the sequential execution (Algorithm 3.1)

$$a = \sigma(n) \tag{3.94}$$

The schedule is admissible if the sequential execution (Algorithm 3.1) does not deadlock and the number of tokens in the SDFBs remains non-negative and bounded. The schedule has a period $R \in \mathbb{N}$

$$\sigma(n+R) = \sigma(n) \tag{3.95}$$

The topology matrix (Definition 3.16) is used to define the repetition vector (Definition 3.18). The repetition vector is used for the PASS calculation (Definition 3.2). Necessary and sufficient conditions for the existence of a PASS [35] are based on the topology matrix, the repetition vector and Algorithm 3.2. These conditions will be referred to in the next section when the validity of the proposed CSM is demonstrated.

Definition 3.16 (Topology Matrix)**.** The topology matrix $\mathbf{\Gamma} \in \mathbb{N}_0{}^{N^{\mathrm{B}} \times N_{\mathrm{A}}}$ of an SDF graph (Definition 3.2) is defined element by element as

$$\Gamma_{ba} = \begin{cases} -\mathrm{c}_a(\imath) + \mathrm{p}_a(o), & (a,\imath) = \mathrm{dst}(b), \quad (a,o) = \mathrm{src}(b) \\ -\mathrm{c}_a(\imath), & (a,\imath) = \mathrm{dst}(b), \quad (\hat{a},\hat{o}) = \mathrm{src}(b), \quad a \neq \hat{a} \\ \ \mathrm{p}_a(0), & (a,o) = \mathrm{src}(b), \quad (\breve{a},\breve{\imath}) = \mathrm{dst}(b), \quad a \neq \hat{a} \end{cases} \tag{3.96}$$

The topology matrix encodes the production and consumption rates of the SDFG. Its rank can be used to check whether the rates are consistent [28], i.e. whether the number of tokens in the SDFBs is limited during execution. The topology matrix can be used to calculate the number of tokens in the SDFBs during sequential execution (Algorithm 3.1). It can be used to reformulate (3.45) and is used when defining the repetition vector.

Definition 3.17 (Greatest common divisor)**.** Let the prime factorization of $n_i \in \mathbb{N}$ be denoted as

$$n_i = \prod_{j=1} p_j^{n_{ij}} \tag{3.97}$$

The greatest common divisor is

$$\gcd\left(n_1, n_2, \ldots, n_N\right) = \prod_{j=1} p_j^{\min\limits_{1 \leqslant i \leqslant N}(n_{ij})} \tag{3.98}$$

The least common multiple is

$$\mathrm{lcm}\left(n_1, n_2, \ldots, n_N\right) = \prod_{j=1} p_j^{\max\limits_{1 \leqslant i \leqslant N}(n_{ij})} \tag{3.99}$$

In the definition above, the fundamental theorem arithmetic is used, which states that all positive integers can be represented in the form of a prime factorization (3.97). The proof and further details on this theorem are available in [67]. Such factorization is used to simplify the notation of the following definitions and theorems. In [35] used the terminology of smallest positive integer vector to describe the repetition vector defined next. In the next definition the expression (3.102) is used to precisely describe the term of smallest positive integer vector.

Definition 3.18 (Repetition Vector)**.** Let $r_a \in \mathbb{N}$ denote the number of times the SDFA with label $a \in I^{\mathrm{A}}$ has repeated its calculation in an iteration. The repetition vector

$$\mathbf{r} = \begin{bmatrix} r_1 & r_2 & \ldots & r_{N^{\mathrm{A}}} \end{bmatrix}^T \tag{3.100}$$

can be calculated by finding the positive integer vector in the null-space of the topology matrix

$$\boldsymbol{\Gamma}\,\mathbf{r} = \mathbf{0} \tag{3.101}$$

such that

$$\gcd_{1\leqslant a\leqslant N_{\mathrm{A}}} \big(r_a\big) = 1 \tag{3.102}$$

The period of a PASS is

$$R = \left(\sum_{a=1}^{N_{\mathrm{A}}} r_a\right) \tag{3.103}$$

The repetition vector indicates the number of SDFA invocations (Algorithm 3.1) after which the number of tokens in the SDFG is equal to the starting number

$$\mathrm{d}\,(2R) = \mathrm{d}_0 \tag{3.104}$$

This statement can be verified with the help of Lemma 3.8.

Algorithm 3.2 Algorithm to find a periodic adimissible sequential schedule (PASS)

```
Require: $G$, $\mathbf{r}$
  $\omega\,(0) := \omega_0, \quad \mathrm{d}\,(0) := \mathrm{d}_0, \quad n := 0, \quad \mathbf{r}' = \mathbf{r}$
  for $a \in I^{\mathrm{A}}$ do
      $k_a := 0$
  repeat
      if $\nexists a \in I^{\mathrm{A}}.\ \exists b \in I^{\mathrm{B}}. \quad {r_a}' > 0, \quad \mathrm{d}(2n,b) < \mathrm{c}_a(\iota), \quad (a,\iota) = \mathrm{dst}(b)$ then
          raise deadlock
      $a := \mathrm{choose}\,(G, \mathrm{d}(2n), \mathbf{r}, n), \quad \begin{array}{l} a \in I^{\mathrm{A}}, \quad {r_a}' > 0, \\ \forall \iota \in I_{\mathscr{U}_a}. \quad (a,\iota) = \mathrm{dst}(b) \Rightarrow \mathrm{d}(2n,b) \geqslant \mathrm{c}_a(\iota) \end{array}$
      $\sigma\,(n+1) := a$
      ${r_a}' := {r_a}' - 1$
      $\big(\mathrm{d}(2n+1), \omega(2n+1), \xi_a(k_a)\big) := \mathrm{consume}\big(\mathrm{d}(2n), \omega(2n), a\big)$
      $\gamma_a(k_a) := \mathrm{calculate}_a\big(\xi_a(k_a)\big)$
      $(\mathrm{d}(2n+2), \omega(2n+2)) := \mathrm{produce}\,(\mathrm{d}(2n+1), \omega(2n+1), a, \gamma_a(k_a))$
      $n := n+1$
      $k_a := k_a + 1$
  until $\forall a \in I^{\mathrm{A}}. \quad m'_a(n) = 0$
```

A modification of Algorithm 3.1 presented in Algorithm 3.2 can be used to find a PASS [35] if one exists. A deadlock occurs when no SDFA can be triggered, i.e. no SDFA can meet (3.35). It is important to note that the initial number of tokens, consumption and production rates are sufficient to find a PASS. It is not necessary to execute the calculation. If the only goal is to find a PASS, Algorithm 3.2 can be performed more efficiently by omitting the calculation. On the other hand, Algorithm 3.2 can be used to execute an SDFG without calculating the schedule.

Example 3.19 (Periodic adimissible sequential schedule)**.** This examples shows how to calculate a PASS of the SDFG presented in Example 3.3. The topology matrix for this SDFG is

$$\mathbf{\Gamma} = \begin{bmatrix} 2 & -1 \\ -2 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{3.105}$$

The repetition vector for this SDFG is

$$\mathbf{r} = \begin{bmatrix} 1 & 2 \end{bmatrix}^T \tag{3.106}$$

and hence the period of a schedule is $R = 3$. The following PASS is obtained by Algorithm 3.2

$$\sigma(n) = \begin{cases} 1, & k = 1 \\ 2, & k \in \{2,3\} \\ \sigma(n-R), & otherwise \end{cases} \tag{3.107}$$

Regardless of the implementation of procedure choose, this is the only possible schedule for the given example.

Example 3.20 (Behavior of an SDFG)**.** The PASS for the SDFG presented in Example 3.3 has been calculated in Example 3.19. The goal of this example is to calculate the token values and present the behavior of the SDFG

$$\begin{bmatrix} \omega(mR,2,1) \\ \omega(mR,2,2) \\ \omega(mR,3,1) \\ \omega(mR,4,1) \end{bmatrix} = \begin{bmatrix} \frac{1}{58} & \frac{1}{87} & \frac{1}{145} & \frac{1}{31} \\ \frac{2497}{465682} & \frac{3760}{1097679} & \frac{6112}{2162095} & \frac{1}{1271} \\ \frac{1}{17} & \frac{1}{19} & \frac{1}{23} & 0 \\ \frac{89}{21238} & \frac{134}{50061} & \frac{218}{98605} & \frac{1}{1681} \end{bmatrix} \begin{bmatrix} \omega((m-1)R,2,1) \\ \omega((m-1)R,2,2) \\ \omega((m-1)R,3,1) \\ \omega((m-1)R,4,1) \end{bmatrix} \tag{3.108}$$

The token values remain the same regardless of the algorithm used to execute the SDFG. However, it will be shown in the next example that the dynamics can change by changing the initial tokens.

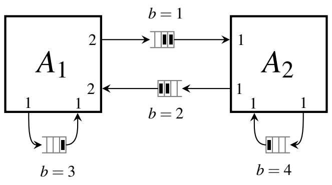


Figure 3.2: The SDFG described in Example 3.21.

Example 3.21 (Effects of initial tokens)**.** This example shows how changing the initial tokens can affect the calculated PASS and system dynamics. The SDFG modified by redistributing its initial tokens is shown in Figure 3.2. The number of initial tokens d_0 is given by

$$\mathrm{d}_0(a,\iota) = \begin{cases} 2, & (a,\iota) = (1,1) \\ 1, & (a,\iota) = (1,2) \\ 2, & (a,\iota) = (2,1) \\ 1, & (a,\iota) = (2,2) \end{cases} \tag{3.109}$$

The repetition vector (3.106) is calculated in Example 3.19. The repetition vector does not change if initial tokens change. Algorithm 3.2 can find three different PASSes depending on the function choose

$$\begin{aligned} \sigma_1(n) &= \begin{cases} 1, & k = 1 \\ 2, & k \in \{2,3\} \\ \sigma(n-R), & otherwise \end{cases} \\ \sigma_2(n) &= \begin{cases} 1, & k = 2 \\ 2, & k \in \{1,3\} \\ \sigma(n-R), & otherwise \end{cases} \\ \sigma_3(n) &= \begin{cases} 2, & k \in \{1,2\} \\ 1, & k = 3 \\ \sigma(n-R), & otherwise \end{cases} \end{aligned} \tag{3.110}$$

Since SDFG is a determinate MOC (Theorem 3.10, Theorem 3.14), the behavior of the SDFG is same regardless of the PASS used for its sequential execution

$$\begin{bmatrix} \omega(mR,1,1) \\ \omega(mR,1,2) \\ \omega(mR,2,1) \\ \omega(mR,2,2) \\ \omega(mR,3,1) \\ \omega(mR,4,1) \end{bmatrix} = \begin{bmatrix} 0 & 0 & \frac{1}{2} & \frac{1}{3} & \frac{1}{5} & 0 \\ 0 & 0 & \frac{1}{7} & \frac{1}{11} & \frac{1}{13} & 0 \\ \frac{1}{29} & 0 & 0 & 0 & 0 & \frac{1}{31} \\ \frac{1}{1147} & \frac{1}{29} & 0 & 0 & 0 & \frac{1}{1271} \\ 0 & 0 & \frac{1}{17} & \frac{1}{19} & \frac{1}{23} & 0 \\ \frac{1}{1517} & \frac{1}{37} & 0 & 0 & 0 & \frac{1}{1681} \end{bmatrix} \begin{bmatrix} \omega((m-1)R,1,1) \\ \omega((m-1)R,1,2) \\ \omega((m-1)R,2,1) \\ \omega((m-1)R,2,2) \\ \omega((m-1)R,3,1) \\ \omega((m-1)R,4,1) \end{bmatrix} \tag{3.111}$$

However, the difference in the order of the discrete systems (3.108) and (3.111) shows the difference in behavior due to change in initial tokens.

Definition 3.22 (SDF iteration)**.** The k^{th} SDF iteration (SDFI) of an SDFG (Definition 3.2) is the set of SDFA invocations in a single period of a PASS (Definition 3.15)

$$iteration(G,k) = \{(a,k_a) \;:\; (a,k_a) \in I^{\mathrm{A}} \times \mathbb{N}, \quad (k-1)\,r_a < k_a \leqslant k\,r_a\} \tag{3.112}$$

The function $iteration : \mathcal{G} \times \mathbb{N} \to 2^{\mathbb{N}\times\mathbb{N}}$ is used to obtain the SDFI.

An SDFI is the basis for the multi-processor algorithm presented in Chapter 4. The previous definition enables the creation of a meaningful list of tasks and their dependencies for multi-processor scheduling algorithms [36]. Since SDF is a determinate MOC, this section describes important tools to verify whether rates of an SDFG are consistent and whether its execution will deadlock.

3.4 Co-simulation network wrapper

This section presents how to specify a CSM (Figure 1.1) using an SDFG. Definition 3.25 shows how a CSN (Definition 2.14) can be wrapped with an SDFG (Definition 3.2). The presented co-simulation network wrapper (CSW) ensures that the repetition vector can be calculated (Theorem 3.28). As a consequence, the CSW has consistent consumption and production rates, i.e. it meets a necessary condition for existence of a PASS [35]. The CSW also ensures that the simulated time increment is consistent for all CSLs involved in the co-simulation (Theorem 3.29).

Definition 3.23 shows how a CSL (Definition 2.13) can be wrapped with an SDFA (Definition 3.1). Such a wrapper is called a simulator. A simulator can easily be used to wrap a CODESUB (Definition 2.1) directly. However, due to the current popularity of the FMI standard [6], an indirect route was chosen.

Definition 3.23 (Simulator)**.** A simulator is an SDFA (Definition 3.1) that executes a CSL (Definition 2.13)

$$A = \left(I_{\mathscr{U}}, \mathscr{U}, I_{\mathscr{Y}}, \mathscr{Y}, \mathrm{c}, \mathrm{p}, \mathrm{calculate}\right) = \mathrm{slaveToSimulator}\left(F, h\right) \tag{3.113}$$

where

- $F \in \mathcal{F}$ is a CSL (Definition 2.13),
- $h \in \mathbb{Q}_{>0}$ is the step size of the simulator,
- and slaveToSimulator : $\mathcal{F} \times \mathbb{Q}_{>0} \rightarrow \mathcal{A}$ is the function that constructs the simulator.

Input port labels are

$$I_{\mathscr{U}} = I_{\mathrm{U}} \cup \{N_{\mathrm{U}} + 1\} \tag{3.114}$$

The input port sets are

$$\mathscr{U}(\imath) = \begin{cases} \mathrm{U}(\imath), & \imath \leqslant N_{\mathrm{U}} \\ V, & \imath = N_{\mathscr{U}} = N_{\mathrm{U}} + 1 \end{cases} \tag{3.115}$$

Output port labels are

$$I_{\mathscr{Y}} = I_{\mathrm{Y}} \cup \{N_{\mathrm{Y}} + 1\} \tag{3.116}$$

The output port sets are

$$\mathscr{Y}(l) = \begin{cases} \mathrm{Y}(l), & l \leqslant N_{\mathrm{U}} \\ V, & l = N_{\mathscr{Y}} = N_{\mathrm{Y}} + 1 \end{cases} \tag{3.117}$$

The consumption rates are

$$\mathrm{c}(\imath) = 1, \quad \imath \in I_{\mathscr{U}} \tag{3.118}$$

The production rates are

$$\mathrm{p}(o) = 1, \quad o \in I_{\mathscr{Y}} \tag{3.119}$$

The function calculate (3.2) assigns input tokens (2.57) as input values of the slave

$$u(k, \imath) = \xi\left(k, \imath, 1\right), \quad \imath \leqslant N_{\mathrm{U}} \tag{3.120}$$

and the state of the slave

$$v(k-1) = \xi\left(k, N_{\mathscr{U}}, 1\right) \tag{3.121}$$

The calculation updates the state of the slave according to (2.58)

$$v(k) = \mathrm{doStep}\left(v^{N_{\mathrm{U}}}(k), h\right) \tag{3.122}$$

and then generates output tokens

$$\gamma(k,o,1) = \begin{cases} \mathrm{get}\big(o, v(k)\big), & o \leqslant N_{\mathrm{Y}} \\ v(k), & o = N_{\mathscr{Y}} \end{cases} \tag{3.123}$$

A simulator is a stateless component (3.140), while the wrapped CSL has a state. The CSL state is modeled by the self-loop (3.8). The consequence of the simulator self-loop can be seen in the definition of simulator input value sets (3.115) and output value sets (3.117), as well as in the CSW constraints (3.136) and (3.137).

Definition 3.24 (Rate converter)**.** A rate converter is an SDFA (Definition 3.1)

$$A = (I_{\mathscr{U}}, \mathscr{U}, I_{\mathscr{Y}}, \mathscr{Y}, \mathrm{c}, \mathrm{p}, \mathrm{calculate}) = \mathrm{converter}\,(c, p) \tag{3.124}$$

where

- $A \in \mathcal{A}$ is the rate converter,
- $c \in \mathbb{N}$ is the consumption rate,
- $p \in \mathbb{N}$ is the production rate,
- and converter : $\mathbb{N} \times \mathbb{N} \to \mathcal{A}$ is the function that constructs the rate converter.

The rate converter has two input and two output ports

$$I_{\mathscr{U}} = \{1,2\}, \quad I_{\mathscr{Y}} = \{1,2\} \tag{3.125}$$

The set of values consumed by the first input port is the same as that generated by the first output port

$$\mathscr{U}\,(1) = \mathscr{Y}\,(1) \tag{3.126}$$

The consumption rates of the rate converter are

$$\mathrm{c}(\iota) = \begin{cases} c, & \iota = 1 \\ 1, & \iota = 2 \end{cases} \tag{3.127}$$

The production rates of the rate converter are

$$\mathrm{p}\,(l) = \begin{cases} p, & l = 1 \\ 1, & l = 2 \end{cases} \tag{3.128}$$

The function calculate (3.2) generates output tokens by resampling input tokens

$$\gamma(k,1,\theta) = \xi\left(k, 1, \left\lceil \frac{\theta\, c}{p} \right\rceil\right) \tag{3.129}$$

Two tasks of the rate converter are rate conversion (3.129) and solving the connection equation (2.63). Resampling tokens (3.129) ensures that a co-simulation network wrapper (CSW) defined next has consistent rates (Theorem 3.28) with a uniform simulated time increment (Theorem 3.29). When two co-simulation slaves (CSLs) $F_{\hat{\imath}}$ and $F_{\check{\imath}}$ are connected (2.62), the following definition of the CSW ensures that their respective simulators are connected. There is a rate converter between each pair of connected ports.

This thesis limits the capabilities of CSLs to use ZOH for reconstruction of input (2.59) and output signals (2.60). This was done to simplify the description of the co-simulation. As already stated in Section 2.3, higher order extrapolation methods are not used in this thesis. This simplification enables the rate conversion (3.129). Such a rate conversion is a stateless calculation. The input port set and the output port set involved in the self-loop are not defined (3.126) because the self-loop is not involved in the calculation. A stateful calculation can be modeled with an additional buffer similar to the case of simulators (Definition 3.23). It would enable the support of more advanced coupling elements such as [68]. In addition, rate converters have exactly one input and one output port. An additional ports can be added to support coupling elements such as [69]. Both generalizations are avoided in this thesis to simplify the notation.

Definition 3.25 (Co-simulation network wrapper)**.** A co-simulation network wrapper (CSW) is an SDFG (Definition 3.2)

$$G = \left(I^{\mathrm{A}}, \mathrm{A}, I^{\mathrm{B}}, \mathrm{src}, \mathrm{dst}, \mathrm{d}_0, \omega_0\right) = \mathrm{networkToDataflow}(\mathcal{C}, \mathrm{h}, \mathrm{d}_0, \omega_0) \tag{3.130}$$

where

- $G \in \mathcal{G}$ is the CSW
- $C \in \mathcal{C}$ is the CSN (Definition 2.61),
- $\mathrm{h} : \ I_{\mathrm{F}} \to \mathbb{Q}_{>0}$ is the function assigning step-sizes of the simulators,
- $\mathrm{d}\ _0 : I^{\mathrm{B}} \to \mathbb{N}_0$ is the number of initial tokens (Definition 3.2),
- $\omega_0 : I^{\mathrm{B}} \to \mathcal{Z}^{\mathbb{N}}$ are the values of initial tokens (Definition 3.2),
- and $\mathrm{networkToDataflow} : \ \mathcal{C} \times (I_{\mathrm{F}} \to \mathbb{Q}_{>0}) \times \left(I^{\mathrm{B}} \to \mathbb{N}_0\right) \times \left(I^{\mathrm{B}} \to \mathcal{Z}^{\mathbb{N}}\right) \to \mathcal{G}$ is the function that constructs the CSW.

The SDFA labels are

$$I^{\mathrm{A}} = \left\{ a \ : \ a \in \mathbb{N}, \quad a \leqslant N_{\mathrm{F}} + \sum_{i=1}^{N_{\mathrm{F}}} N_{\mathrm{U}_i} \right\} \tag{3.131}$$

The first N_F SDFAs are simulators and the rest are rate converters

$$\mathrm{A}(a) = \begin{cases} \mathrm{slaveToSimulator}\left(\mathrm{F}\left(i\right), \mathrm{h}\left(i\right)\right), & a = i, \quad i \in I_\mathrm{F} \\ \mathrm{converter}\left(\frac{\mathrm{lcm}(d_{\hat{\imath}}\eta_{\breve{i}},\, d_{\breve{i}}\eta_{\hat{\imath}})}{d_{\breve{i}}\eta_{\hat{\imath}}}, \frac{\mathrm{lcm}(d_{\hat{\imath}}\eta_{\breve{i}},\, d_{\breve{i}}\eta_{\hat{\imath}})}{d_{\hat{\imath}}\eta_{\breve{i}}}\right), & \begin{array}{l} a = N_\mathrm{F} + \breve{\imath} + \sum\limits_{j=1}^{\breve{i}-1} N_{\mathrm{U}_j}, \\ (\hat{\imath}, \hat{o}) = \mathrm{L}\left(\breve{i}, \breve{\imath}\right), \\ \mathrm{h}\left(\hat{\imath}\right) = \frac{\eta_{\hat{\imath}}}{d_{\hat{\imath}}}, \quad \mathrm{h}\left(\breve{i}\right) = \frac{\eta_{\breve{i}}}{d_{\breve{i}}} \\ \gcd(\eta_{\hat{\imath}}, d_{\hat{\imath}}) = 1, \quad \gcd(\eta_{\breve{i}}, d_{\breve{i}}) = 1 \\ \breve{i} \in I_\mathrm{F}, \quad \breve{\imath} \in I_{\mathrm{U}_i} \end{array} \end{cases} \tag{3.132}$$

The SDFB labels are

$$I^\mathrm{B} = \left\{ b \,:\, b \in \mathbb{N}, \quad b \leqslant N_\mathrm{A} + 2\sum_{i=1}^{N_\mathrm{F}} N_{\mathrm{U}_i} \right\} \tag{3.133}$$

The source ports of the SDFBs are given by

$$\mathrm{src}(b) = \begin{cases} (a, N_{\mathscr{Y}_a}), & b \leqslant N_\mathrm{A}, \quad a = b \\ (\underline{\mathrm{a}}, 1), & b = N_\mathrm{A} + \breve{\imath} + \sum_{j=1}^{\breve{i}-1} N_{\mathrm{U}_j}, \quad \underline{\mathrm{a}} = N_\mathrm{F} + \breve{\imath} + \sum_{j=1}^{\breve{i}-1} N_{\mathrm{U}_j}, \quad \breve{i} \in I_\mathrm{F}, \quad \breve{\imath} \in I_{\mathrm{U}_{\breve{i}}} \\ (\hat{\imath}, \hat{o}), & b = N_\mathrm{A} + \breve{\imath} + \sum_{j=1}^{\breve{i}-1} N_{\mathrm{U}_j} + \sum_{j=1}^{N_\mathrm{F}} N_{\mathrm{U}_j}, \quad (\hat{\imath}, \hat{o}) = \mathrm{L}\left(\breve{i}, \breve{\imath}\right), \quad \breve{i} \in I_\mathrm{F}, \quad \breve{\imath} \in I_{\mathrm{U}_{\breve{i}}} \end{cases} \tag{3.134}$$

The destination ports of the SDFBs are given by

$$\mathrm{dst}(b) = \begin{cases} (a, N_{\mathscr{U}_a}), & b \leqslant N_\mathrm{A}, \quad a = b \\ (\breve{i}, \breve{\imath}), & b = N_\mathrm{A} + \breve{\imath} + \sum_{j=1}^{\breve{i}-1} N_{\mathrm{U}_j}, \quad \breve{i} \in I_\mathrm{F}, \quad \breve{\imath} \in I_{\mathrm{U}_i} \\ (\underline{\mathrm{a}}, 1), & \begin{array}{l} b = N_\mathrm{A} + \breve{\imath} + \sum_{j=1}^{\breve{i}-1} N_{\mathrm{U}_j} + \sum_{j=1}^{N_\mathrm{F}} N_{\mathrm{U}_j}, \\ \underline{\mathrm{a}} = N_\mathrm{F} + \breve{\imath} + \sum_{j=1}^{\breve{i}-1} N_{\mathrm{U}_j}, \quad \breve{i} \in I_\mathrm{F}, \quad \imath \in I_{\mathrm{U}_i} \end{array} \end{cases} \tag{3.135}$$

The number of initial tokens in the SDFBs of a CSW is constrained by

$$\mathrm{d}_0(b) = 1, \quad b \leqslant N_\mathrm{A} \tag{3.136}$$

The initial tokens in the SDFBs containing the internal states of the CSLs are

$$\omega_0\left(b\right) = v_{0i}, \quad b \leqslant N_\mathrm{F}, \quad i = b \tag{3.137}$$

Definition 3.23, Definition 3.24 and Definition 3.25 introduce a MOC for the simulation of a CSN (Definition 2.14). The CSW decouples the execution platform from the definition of the behavior. This means that Algorithm 3.1 can be used to execute the generated CSW as well as the multiprocessor algorithm presented in the next chapter.

A CSW contains

$$N_{\mathrm{A}} = N_{\mathrm{F}} + \sum_{i=1}^{N_{\mathrm{F}}} N_{\mathrm{U}_i} \tag{3.138}$$

SDFAs (3.132), of which N_{F} are simulators and $\sum_{i=1}^{N_{\mathrm{F}}} N_{\mathrm{U}_i}$ are rate converters. There is one simulator for each CSL in CSN C and one rate converter for each input port of each CSL. A CSW contains

$$N^{\mathrm{B}} = N_{\mathrm{A}} + 2\sum_{i=1}^{N_{\mathrm{F}}} N_{\mathrm{U}_i} \tag{3.139}$$

SDFB (3.133), of which N_{F} enclose the internal states of the slaves and $2\sum_{i=1}^{N_{\mathrm{F}}} N_{\mathrm{U}_i}$ are used to connect simulators and rate converters. The source (3.134) and destination (3.135) of SDFBs that enclose the internal state of an SDFA are the same

$$(a, N_{\mathscr{Y}_a}) = \mathrm{src}(b), \quad (a, N_{\mathscr{U}_a}) = \mathrm{dst}(b), \quad b \leqslant N_{\mathrm{A}}, \quad a = b \tag{3.140}$$

These buffers initially only contain one token (3.136) with the initial state of the respective slaves (3.137). Such self-loops impose precedence constraints on the parallel execution of the graph presented in Chapter 4. A multiprocessor schedule can lead to incorrect results without having information about stateful calculations [28]. There are two SDFBs connected to each rate converter, one at the input port and one at the output port of the converter. A rate converter with the two connected buffers solves the connection equation (2.63). The equations (3.134) and (3.135) imply that there is an input and an output buffer for each rate converter

$$\begin{gathered}
(\hat{\imath}, \hat{o}) = \mathrm{src}\left(\hat{b}\right), \quad \mathrm{dst}\left(\hat{b}\right) = (\underline{\mathrm{a}}, 1), \quad (\hat{\imath}, \hat{o}) = \mathrm{src}\left(\breve{b}\right), \quad \mathrm{dst}\left(\breve{b}\right) = (\breve{\imath}, \breve{\mathrm{\imath}}), \\
\hat{b} = N_{\mathrm{F}} + \breve{\imath} + \sum_{j=1}^{\breve{\imath}-1} N_{\mathrm{U}_j} + \sum_{j=1}^{N_{\mathrm{F}}} N_{\mathrm{U}_j}, \quad \breve{b} = N_{\mathrm{F}} + \breve{\imath} + \sum_{j=1}^{\breve{\imath}-1} N_{\mathrm{U}_j} \\
\underline{\mathrm{a}} = N_{\mathrm{F}} + \breve{\imath} + \sum_{j=1}^{\breve{\imath}-1} N_{\mathrm{U}_j}, \quad (\hat{\imath}, \hat{o}) = \mathrm{L}(\breve{\imath}, \breve{\mathrm{\imath}}), \quad \breve{\imath} \in I_{\mathrm{F}}, \quad \breve{\mathrm{\imath}} \in I_{\mathrm{U}_i}
\end{gathered} \tag{3.141}$$

The equations (3.138)-(3.141) are used to provide additional insight into the definitions in this section.

Theorem 3.28 shows that a CSW defined in this way has consistent consumption and production rates. It shows that it is possible to calculate the repetition vector of a CSW (Definition 3.25).

Definition 3.26 (Generalized step sizes)**.** Let $\mathrm{h}' : I^{\mathrm{A}} \to \mathbb{Q}_{>0}$ be the function selecting generalized step sizes of SDFAs induced by a CSW (Definition 3.25) defined as

$$\mathrm{h}'(a) = \begin{cases} \mathrm{h}(a), & a \leqslant N_\mathrm{F} \\ \frac{\operatorname{lcm}(d_{\hat{a}}\,\eta_{\check{a}},\ d_{\check{a}}\,\eta_{\hat{a}})}{d_{\hat{a}}\,d_{\check{a}}}, & a > N_\mathrm{F}, \quad \begin{matrix} (\hat{a},\hat{o}) = \mathrm{src}(\hat{b}), & (a,1) = \mathrm{dst}(\hat{b}) \\ (a,1) = \mathrm{src}(\check{b}), & (\check{a},\check{\imath}) = \mathrm{dst}(\check{b}) \end{matrix} \end{cases} \tag{3.142}$$

where the step sizes of the simulators are

$$\mathrm{h}(a) = \tfrac{\eta_a}{d_a} \in \mathbb{Q}_{>0} \tag{3.143}$$

and respective numerators and denominators are relatively prime

$$\gcd(\eta_a, d_a) = 1 \tag{3.144}$$

Lemma 3.27 (Inverted step sizes)**.** *The vector of inverted generalized step sizes (Definition 3.26) is in the kernel of the topology matrix (Definition 3.16) generated by a CSW (Definition 3.25)*

$$\mathbf{\Gamma} \begin{bmatrix} \frac{1}{\mathrm{h}'(1)} & \frac{1}{\mathrm{h}'(2)} & \cdots & \frac{1}{\mathrm{h}'(N_\mathrm{A})} \end{bmatrix}^T = \mathbf{0} \tag{3.145}$$

Proof. Let C be the CSN (Definition 2.14) and L its connection function (2.62). Let

$$G = \mathrm{networkToDataflow}(C, \mathrm{h}, \mathrm{d}_0, \omega_0) \tag{3.146}$$

be the CSW that wraps . Let

$$(\hat{\imath},\hat{o}) = \mathrm{L}(\check{\imath},\check{\imath}), \quad \hat{\imath},\check{\imath} \in I_\mathrm{F}, \quad \hat{o} \in I_{\mathrm{Y}_{\hat{\imath}}}, \quad \check{\imath} \in I_{\mathrm{U}_{\check{\imath}}} \tag{3.147}$$

From (3.134) and (3.135) it follows there exists $\check{b},\hat{b} \in I^\mathrm{B}$ and $\underline{a} \in I^\mathrm{A}$ such that

$$(\hat{\imath},\hat{o}) = \mathrm{src}(\hat{b}) \quad (\underline{a},1) = \mathrm{dst}(\hat{b}) \tag{3.148}$$

and

$$(\underline{a},1) = \mathrm{src}(\check{b}) \quad (\check{\imath},\check{\imath}) = \mathrm{dst}(\check{b}) \tag{3.149}$$

From (3.118), (3.119), (3.124), (3.132) and (3.142) it follows that

$$\mathrm{p}_{\hat{\imath}}(\hat{o})\,\frac{1}{\mathrm{h}(\hat{\imath})} - \mathrm{c}_{\underline{a}}(1)\,\frac{1}{\mathrm{h}'(\underline{a})} = 0 \tag{3.150}$$

and

$$\mathrm{p}_{\underline{a}}(1)\,\frac{1}{\mathrm{h}'(\underline{a})} - \mathrm{c}_{\check{\imath}}(\check{\imath})\,\frac{1}{\mathrm{h}(\check{\imath})} = 0 \tag{3.151}$$

The rows of the topology matrix (Definition 3.16) are formed for each buffer in the CSW. From (3.5), (3.6), (3.134) and (3.135) it follows that for the buffers in a self-loop

$$\mathrm{p}_a\left(N_{\mathscr{Y}_a}\right) \frac{1}{\mathrm{h}'(a)} - \mathrm{c}_a(N_{\mathscr{U}_a}) \frac{1}{\mathrm{h}'(a)} = 0 \tag{3.152}$$

all of the elements of the vector

$$\mathbf{\Gamma} \begin{bmatrix} \frac{1}{\mathrm{h}'(1)} & \frac{1}{\mathrm{h}'(2)} & \cdots & \frac{1}{\mathrm{h}'(N_\mathrm{A})} \end{bmatrix}^T \tag{3.153}$$

can be expressed either by (3.150), (3.151) or (3.152). □

Theorem 3.28 (Repetition vector)**.** *The repetition vector* $\mathbf{r} \in \mathbb{N}^{N_\mathrm{A}}$ *(Definition 3.18) of a CSW (Definition 3.25) can be calculated by*

$$\mathbf{r} = \frac{\mathrm{lcm}\left(\eta'_1, \eta'_2, \ldots, \eta'_{N_\mathrm{A}}\right)}{\mathrm{gcd}\left(d'_1, d'_2, \ldots, d'_{N_\mathrm{A}}\right)} \begin{bmatrix} \frac{1}{\mathrm{h}'(1)} & \frac{1}{\mathrm{h}'(2)} & \cdots & \frac{1}{\mathrm{h}'(N_\mathrm{A})} \end{bmatrix}^T \tag{3.154}$$

where the generalized step sizes of the SDFAs are

$$\mathrm{h}'(a) = \tfrac{\eta'_a}{d'_a} \in \mathbb{Q}_{>0} \tag{3.155}$$

and respective numerators and denominators are relatively prime

$$\mathrm{gcd}(d'_a, \eta'_a) = 1 \tag{3.156}$$

Proof. Lemma 3.27 shows that (3.154) is in the kernel of the topology matrix. From Definition 3.17 it follows that

$$r_a = \frac{\underset{1\leqslant a'\leqslant N_\mathrm{A}}{\mathrm{lcm}} \left(\eta'_{a'}\right)}{\underset{1\leqslant a'\leqslant N_\mathrm{A}}{\mathrm{gcd}} \left(d'_{a'}\right)} \frac{d'_a}{\eta'_a} = \prod_{j=1} p_j^{\max\limits_{1\leqslant a'\leqslant N_\mathrm{A}} (\eta'_{a'j}) - \eta_{aj} + d_{aj} - \min\limits_{1\leqslant a'\leqslant N_\mathrm{A}} (d'_{a'j})} \tag{3.157}$$

Each element of the vector is a positive integer since

$$\max_{1\leqslant a'\leqslant N_\mathrm{A}} (\eta'_{a'j}) - \eta_{aj} \geqslant 0 \tag{3.158}$$

and

$$d_{aj} - \min_{1\leqslant a'\leqslant N_\mathrm{A}} (d'_{a'j}) \geqslant 0 \tag{3.159}$$

The condition (3.102) holds if

$$\min_{1\leqslant a\leqslant N_\mathrm{A}} \Big(\max_{1\leqslant a'\leqslant N_\mathrm{A}} (\eta'_{a'j}) - \eta_{aj} + d_{aj} - \min_{1\leqslant a'\leqslant N_\mathrm{A}} (d'_{a'j}) \Big) = 0 \tag{3.160}$$

for all $j \in \mathbb{N}$. From (3.156) it follows that

$$\min(\eta'_{aj}, d'_{aj}) = 0, \quad 1 \leqslant a \leqslant N_\mathrm{A} \tag{3.161}$$

If there exists $a \in I^\mathrm{A}$ such that

$$\max_{1 \leqslant a' \leqslant N_\mathrm{A}} (\eta'_{a'j}) = \eta'_{aj} > 0 \tag{3.162}$$

then

$$d'_{aj} = 0, \quad \min_{1 \leqslant a' \leqslant N_\mathrm{A}} (d'_{a'j}) = 0 \tag{3.163}$$

follows from (3.161) and consequently (3.160) holds. If

$$\max_{1 \leqslant a' \leqslant N_\mathrm{A}} (\eta'_{a'j}) = 0 \tag{3.164}$$

then there exists $a \in I^\mathrm{A}$ such that

$$d'_{aj} = \min_{1 \leqslant a' \leqslant N_\mathrm{A}} (d'_{a'j}), \quad n'_{aj} = 0 \tag{3.165}$$

and consequently (3.160) holds. □

Theorem 3.29 (SDFI time increment)**.** *Let C be a CSN (Definition 2.14) and*

$$G = \text{networkToDataflow}(C, \mathrm{h}, \mathrm{d}_0, \omega_0) \tag{3.166}$$

its CSW (Definition 3.25). The simulated time increment in the k^{th} SDFI (Definition 3.22) of the i^{th} simulator is

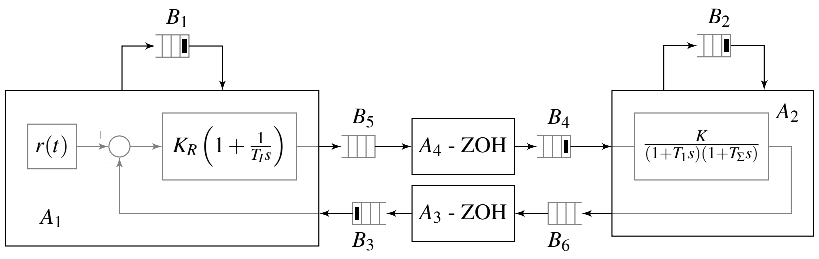
$$H = t_i(k\, r_i) - t_i((k-1)\, r_i) = r_a\, \mathrm{h}(i) = \frac{\operatorname{lcm}\left(\eta'_1, \eta'_2, \ldots, \eta'_{N_\mathrm{A}}\right)}{\gcd\left(d'_1, d'_2, \ldots, d'_{N_\mathrm{A}}\right)} \tag{3.167}$$

Proof. From (2.56) and (3.122) it follows that the simulated time after the n^{th} execution of the i^{th} simulator is

$$t_i(n) = n\, \mathrm{h}(i) \tag{3.168}$$

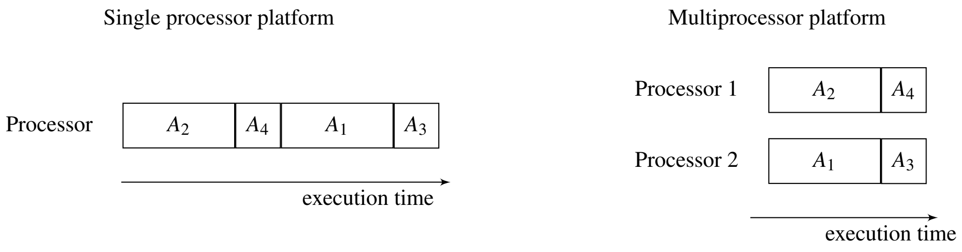
The expression (3.167) follows from the expression for the repetition vector (3.154) and (3.168). □

Theorem 3.29 shows that each CSL in a CSW has a uniform simulated time increment. The connection equations (2.63) give an expression for input and output signals at the same simulated time. If the simulated time of the connected CSLs is not increased uniformly, the



B_1
B_2
A_2
B_5
B_4
$r(t)$
+
−
$K_R\left(1+\frac{1}{T_I s}\right)$
A_4 - ZOH
$\frac{K}{(1+T_1 s)(1+T_\Sigma s)}$
A_1
A_3 - ZOH
B_3
B_6

(a) The SDFG consists of 4 SDFAs (2 simulators and 2 rate converters) and 6 SDFBs.



(b) A scheduling algorithm can optimize the execution time if more processors are available for the running the above SDFG.

Figure 3.3: The SDFG presented in Example 3.30 and its execution on two different execution platforms.

numerical solution of the connection equations is likely to have a large error.

Example 3.30 (Control loop)**.** This example presents the CSW (Definition 3.25) for the CSN introduced in Example 2.18. The control loop CSN C is constructed with (2.75). The step-sizes of the simulators are

$$\mathrm{h}(a) = \begin{cases} \frac{1}{2}, & a = 1 \\ \frac{1}{2}, & a = 2 \end{cases} \tag{3.169}$$

The number of initial tokens of the CSW is

$$\mathrm{d}_0\left(b\right) = \begin{cases} 1, & b \leqslant 4 \\ 0, & otherwise \end{cases} \tag{3.170}$$

The values of initial tokens of the CSW are

$$\omega_0\left(b,\theta\right) = \begin{cases} v_{01}, & b = 1, \quad \theta = 1 \\ v_{02}, & b = 2, \quad \theta = 1 \\ y_{11}(0.5), & b = 3, \quad \theta = 1 \\ y_{21}(0.5), & b = 4, \quad \theta = 1 \end{cases} \tag{3.171}$$

The CSW is

$$G = \mathrm{networkToDataflow}(C, \mathrm{h}, \mathrm{d}_0, \omega_0) \tag{3.172}$$

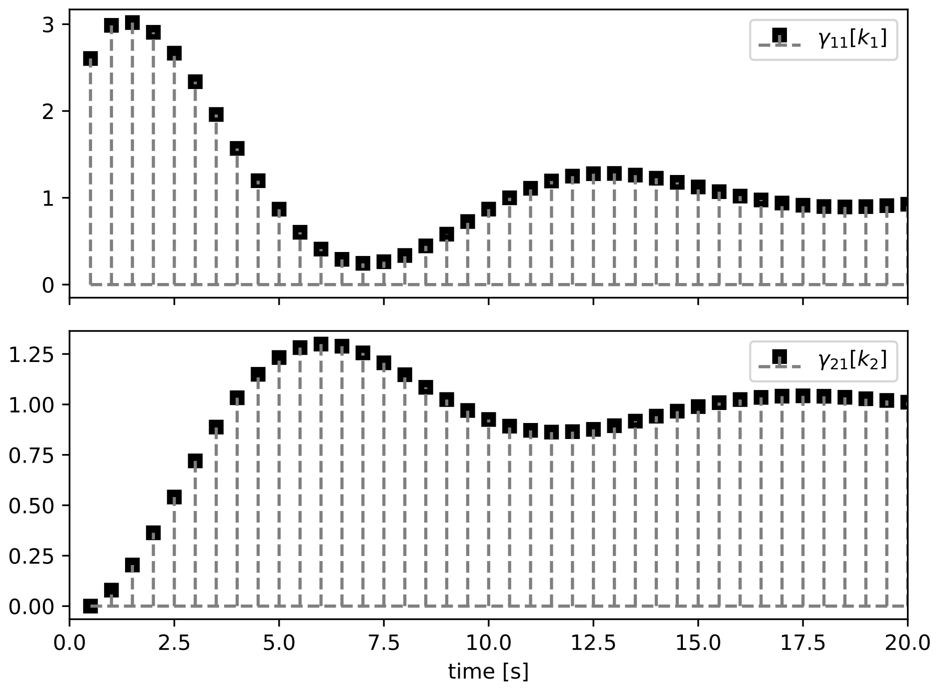


Figure 3.4: The token values obtained by the execution of the CSW presented in Example 3.30.

The Python procedure `sdf4sim.example.control.gauss_jacobi_csw_run` available at [44] runs the CSW (3.172).

Example 3.30 shows a non-iterative version of a Jacobi CSM [31] as a CSW. It is interesting that changing the number of initial tokens (3.170) to

$$\mathrm{d}_0(b) = \begin{cases} 1, & b \leqslant 3 \\ 0, & otherwise \end{cases} \tag{3.173}$$

makes modified CSW a non-iterative version of a Gauss-Seidel CSM [31].

An advantage of such a view is that it links the MOC to the work that has researched its implementation [36]. If a non-iterative co-simulation is represented as a CSW, execution can be statically planned on both single and multiprocessor platforms [35].

Chapter 4

Execution platform

This chapter presents an idealistic model for the execution platform, the infinite processor execution platform (IPEP) in Definition 4.1. Such a platform enables the analysis of a co-simulation schedule. Scheduling is a decision-making process that deals with the allocation of processors to tasks [70]. The tasks in co-simulation are single executions of the calculation functions of simulators and rate converters. The connection to the scheduling research area can be observed by comparing the IPEP (Definition 4.1) and the project defined in [70]. The two platforms are functionally the same. Such a platform can be easily generalized to machines in parallel with different speeds or to unrelated machines in parallel [70].

The generalization was avoided for the sake of simplicity, since scheduling is a well-researched topic, both in the case of SDFGs [35, 36] and as a general topic [70, 71]. In [35], authors describe static algorithms for scheduling SDFGs on single and multiprocessor platforms. The algorithm for scheduling an SDFG on a single processor (Algorithm 3.2) is also described in Section 3.3. This chapter introduces an algorithm for scheduling execution on the IPEP (Algorithm 4.2). However, the main objective of this chapter is to provide an analysis of the real-time capabilities of a CSW.

A CSW can be executed in real time if its makespan is less than or equal to its SDFI time increment (Theorem 3.29). The makespan analysis for the IPEP is shown in Section 4.1. This makespan analysis is considered useful in practice because it provides an optimistic estimate of the real-time capability of the CSW. If the CSW cannot run in real time on the IPEP, it cannot run in real time on any other platform.

The makespan analysis presented suggests how to develop a method for calculating the initial CSW tokens. Such a method is presented and analyzed in Section 4.2. Since SDF is a determinate MOC, a CSW created with this selection of tokens can be reused for different execution platforms. A similar approach is demonstrated in [65], where timing information is discarded from a fully static schedule for use in self-timed execution.

The proposed method for selecting initial tokens enables HIL experiments to be modeled using a CSW. There hardware introduces real-time execution constraints for the SDFG. A HIL simulation can be a part of the MBD presented in the last section of this chapter. This makes the proposed method for selecting initial tokens a reasonable default when selecting the MOC for different phases of the MBD.

4.1 Makespan of a synchronous data flow iteration

The SDFAs in an SDFG describe cyclical tasks. The schedule in this chapter is a policy that is implicitly introduced in Algorithm 4.2. This policy states that an SDFA calculation is executed as soon as there are enough tokens in the SDFA's input SDFBs (3.35). Such a policy is possible if the execution platform contains an unlimited[*] number of processors (Definition 4.1). It is important to note that Algorithm 4.2 executes SDF iterations (Definition 3.22) sequentially, but the SDFA calculations within one iteration are performed concurrently if possible. This makes it possible to define the makespan of an SDFI (Definition 4.6).

Definition 4.1 (Infinite processor execution platform)**.** The infinite processor execution platform (IPEP) consists of infinite identical machines. The time it takes to execute a single invocation of the calculation function is specified with the function

$$\mathrm{time} : \mathcal{A} \to \mathbb{R}_{>0} \tag{4.1}$$

Definition 4.1 introduces a platform that corresponds functionally to the project defined in [70]. The problem of minimizing the makespan of a project can be optimally solved [70]. This problem has a known solution, the critical path method commonly used in project management [72]. The next definition describes the list of tasks that are performed repeatedly in Algorithm 4.2.

Algorithm 4.1 SDFA invocation

```
Require: $G \in \mathcal{G}$,   $\mathrm{d}(n') : I^{\mathrm{B}} \to \mathbb{N}_0$,   $\omega(n') : I^{\mathrm{B}} \to \mathcal{U}^{\mathbb{N}}$,   $a \in I^{\mathrm{A}}$
  $\big(\mathrm{d}(n'+1), \omega(n'+1), \xi_a(k_a)\big) := \mathrm{consume}\big(\mathrm{d}(n'), \omega(n'), a\big)$
  $\gamma_a(k_a) := \mathrm{calculate}_a\big(\xi_a(k_a)\big)$
  $\big(\mathrm{d}(n''+1), \omega(n''+1)\big) := \mathrm{produce}\big(\mathrm{d}(n''), \omega(n''), a, \gamma_a(k_a)\big)$
```

Algorithm 4.1 presents a task that is scheduled on a processor. The task consists of the consumption of tokens, the SDFA calculation and the production of tokens. It is a building block that is used in Algorithm 4.2.

[*]or large enough

Algorithm 4.2 Execution of an SDFG on the IPEP

```
Require: $G \in \mathcal{G}$,   a PASS exists
  $\omega(0) := \omega_0$,   $\mathrm{d}(0) := \mathrm{d}_0$
  $n := 0$
  repeat
      Wait for the clock trigger          ▷ This command enables execution in real time.
      $\mathbf{r}' := \mathbf{r}$
      repeat                              ▷ This loop executes SDFIs in a sequence.
          for $a \in \left\{a' \,:\, a' \in I^{\mathrm{A}},\quad r'_{a'} > 0,\quad \forall \iota' \in I_{\mathscr{U}_{a'}}.\, \mathrm{d}(n,b) \geqslant \mathrm{c}_{a'}(\iota'),\ (a',\iota') = \mathrm{dst}(b)\right\}$ do
              Start the execution of the task specified in Algorithm 4.1 for the SDFA labeled $a$
              ▷ Each of the tasks specified by the loop is scheduled simultaneously due to the
                infinite number of processors.
      until $\forall a \in I^{\mathrm{A}}.\quad r'_{a'} = 0$
  until external stop
```

Definition 4.2 (Precedence constraints)**.** The set of precedence constraints for the execution of an SDFG (Definition 3.2) is referred to as

$$\text{precedences} : \mathcal{G} \times \mathbb{N} \times \mathbb{N} \to 2^{\mathbb{N}\times\mathbb{N}} \tag{4.2}$$

The $k^{th}_{\hat{a}}$ invocation of the SDFA labeled $\hat{a}$ is a precedence constraint for the k^{th} invocation of the SDFA labeled a

$$(\hat{a}, k_{\hat{a}}) \in \text{precedences}(G,k,a) \tag{4.3}$$

if there exists $b \in I^{\mathrm{B}}$, $\hat{o} \in I_{\mathscr{Y}_{\hat{a}}}$ and $\iota \in I^{\mathrm{A}}$ such that

$$\mathrm{src}(b) = (\hat{a},\hat{o}), \quad \mathrm{dst}(b) = (a,\iota), \quad \hat{o} \in I_{\mathscr{Y}_{\hat{a}}}, \quad \iota \in I_{\mathscr{U}_a} \tag{4.4}$$

and

$$k_a\ \mathrm{c}_a(\iota) > (k_{\hat{a}} - 1)\ \mathrm{p}_{\hat{a}}(\hat{o}) + \mathrm{d}_0(b) \tag{4.5}$$

The above definition introduces the set of precedence constraints on the invocations of an SDFAs while running an SDFI. The next theorem shows how the precedence constraints affect the execution of the SDFA. A precedence constraint specifies a rule for how calls from connected SDFA invocations should be sequenced.

Theorem 4.3 (Precedence constraints)**.** *Assume that an SDFG (Definition 3.2) is executed with a valid execution (Definition 3.7). Let*

$$k_{\hat{a}} = \mathrm{num}(\text{produce}, \hat{a}, \hat{n}), \quad \mathrm{operation}(\hat{n}) = (\text{produce}, \hat{a}) \tag{4.6}$$

and

$$k_{\breve{a}} = \mathrm{num}\left(\mathrm{consume}, \breve{a}, \breve{n}\right), \quad \mathrm{operation}(\breve{n}) = (\mathrm{consume}, \breve{a}) \tag{4.7}$$

If there exists $b \in I^{\mathrm{B}}$ such that

$$(\hat{a}, \hat{o}) = \mathrm{src}(b), \quad (\breve{a}, \breve{\imath}) = \mathrm{dst}(b), \quad \hat{o} \in I_{\mathscr{Y}_{\hat{a}}}, \quad \breve{\imath} \in I_{\mathscr{U}_{\breve{a}}} \tag{4.8}$$

and

$$k_{\breve{a}}\ \mathrm{c}_{\breve{a}}(\breve{\imath}) > (k_{\hat{a}} - 1)\ \mathrm{p}_{\hat{a}}(\hat{o}) + \mathrm{d}_0(b) \tag{4.9}$$

then

$$\breve{n} > \hat{n} \tag{4.10}$$

Proof. The proof is given by contradiction. Assume that

$$\breve{n} < \hat{n} \tag{4.11}$$

From (3.26) if follows that $\breve{n} \neq \hat{n}$. From (3.30) and (4.7) it follows that

$$\mathrm{num}\left(\mathrm{produce}, \breve{a}, \breve{n} - 1\right) = k_{\breve{a}} - 1 \tag{4.12}$$

From (3.30), (4.6), (4.7) and (4.11) it follows that

$$\mathrm{num}\left(\mathrm{produce}, \hat{a}, \hat{n}\right) = \mathrm{num}\left(\mathrm{produce}, \hat{a}, \breve{n} - 1\right) + \sum_{n=\breve{n}}^{\hat{n}} \mathrm{count}(\mathrm{produce}, a, n) = k_{\hat{a}} \tag{4.13}$$

and consequently

$$\mathrm{num}\left(\mathrm{produce}, \hat{a}, \breve{n} - 1\right) \leqslant k_{\hat{a}} - 1 \tag{4.14}$$

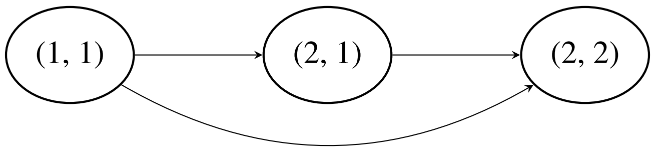
From (4.9), (4.12) and (4.14) it follows that

$$\mathrm{num}\left(\mathrm{consume}, \breve{a}, \breve{n} - 1\right)\ \mathrm{c}_{\breve{a}}(\breve{\imath}) + \mathrm{c}_{a}(\breve{\imath}) > \mathrm{d}_0(b) + \mathrm{num}\left(\mathrm{produce}, \hat{a}, \breve{n} - 1\right)\ \mathrm{p}_{\hat{a}}(\hat{o}) \tag{4.15}$$

This contradicts Lemma 3.9. □

The precedence constraints of an SDFG can be presented in the form of an acyclic precedence graph. This representation can be used to calculate a periodic admissible parallel schedule [35]. The next example is introduced to connect the work in the thesis with the existing research on the SDFG scheduling [36]. This thesis does not attempt to further examine the scheduling problem. The goal of this chapter is to present the procedure for initial token selection in the next section. This section introduces the prerequisites to justify the selection.

Example 4.4 (Acyclic precedence graph)**.** This example shows how an acyclic precedence graph is created from the SDFG shown in Example 3.3. The nodes of an acyclic precedence graph are invocations of SDFAs in a single SDFI (Definition 3.22). There is an edge between two SDFA invocations to which the precedence constraint (4.9) applies.



The graph can be used to execute an SDFG on a multiprocessor platform [35, 36, 65, 73]. The acyclic precedence graph is usually pruned to remove redundant constraints. In this example, the edge from $(1,1)$ to $(2,2)$ can be removed. The pruned acyclic precedence graph can still be used for scheduling the SDFG on a multiprocessor platform.

Example 4.4 shows how to create an acyclic precedence graph [35, 36, 65, 73]. This notation is not used in the rest of the thesis. The example was only used to connect this section to existing research.

Definition 4.5 (Completion time)**.** Assume that an SDFG G (Definition 3.2) is executed on the IPEP (Definition 4.1). The function $T : \mathcal{G} \times \mathbb{N} \times \mathbb{N} \to \mathbb{R}_{>0}$ denotes the completion time of an SDFA invocation

$$T(G,a,k) = \begin{cases} \mathrm{time}\big(\mathrm{A}(a)\big), & \mathrm{precedences}(G,a,k) = \emptyset \\ \mathrm{time}(a) + \max\limits_{(\hat{a},k_{\hat{a}}) \in \mathrm{precedences}(G,a,k)} T(G,\hat{a},k_{\hat{a}}), & \textit{otherwise} \end{cases} \tag{4.16}$$

The above definition of completion time enables max-plus algebra timing analysis on the IPEP [74]. Basic max-plus algebra is used in the proofs shown in the next section.

Definition 4.6 (Makespan of an SDF iteration)**.** Assume that an SDFG (Definition 3.2) is executed with Algorithm 4.2. The makespan of an SDFI (Definition 3.22) is

$$T_{iteration}(G) = \max_{(a,k) \in iteration(G)} T(G,a,k) \tag{4.17}$$

The next section uses the makespan definition to justify the method for selecting the number of initial tokens. The next example gives an indication of how the number of initial tokens affects the makespan of the SDFI.

Example 4.7 (Makespan of an SDF iteration)**.** Let

$$G_1 = (I^{\mathrm{A}}, \mathrm{A}, I^{\mathrm{B}}, \mathrm{src}, \mathrm{dst}, \mathrm{d}_{01}, \omega_{01}) \tag{4.18}$$

denote the SDFG presented in Example 3.3 and

$$G_2 = (I^{\mathrm{A}}, \mathrm{A}, I^{\mathrm{B}}, \mathrm{src}, \mathrm{dst}, \mathrm{d}_{02}, \omega_{02}) \tag{4.19}$$

denote the SDFG presented in Example 3.21. The two SDFGs have the same SDFAs and different initial tokens. Let the execution time of the SDFAs be

$$\mathrm{time}(A) = \begin{cases} 5, & A = \mathrm{A}(1) \\ 3, & A = \mathrm{A}(2) \end{cases} \tag{4.20}$$

If the SDFGs are executed with Algorithm 4.1 then

$$T_{iteration}(G_1) = 11 \tag{4.21}$$

and

$$T_{iteration}(G_2) = 6 \tag{4.22}$$

This example shows that the execution time of an SDFI can be reduced by changing the initial tokens and executing the SDFG on the IPEP. This advantage comes with a compromise. Comparing Example 3.20 and Example 3.21 shows that changing initial tokens changes the behavior of the SDFG.

4.2 Number of initial tokens

This section presents the method for calculating the number of initial tokens (Definition 4.8) based on the simulator step sizes in the CSW. To execute the CSN (Definition 2.14) with the CSW, an engineer must specify the simulator step sizes, the number of initial tokens, and the initial token values. Suppose the number of initial tokens of a CSW is calculated using the method described in Definition 4.8. This section shows that such a CSW does not deadlock (Corollary 4.13) and can run in real time on the IPEP (Theorem 4.17). The proposed method provides a default setting for selecting the number of initial tokens.

Definition 4.8 (Number of initial tokens)**.** Let

- $C \in \mathcal{C}$ be a CSN (Definition 2.14)
- and h : $I_{\mathrm{F}} \to \mathbb{Q}_{>0}$ the simulator step sizes of the CSW (Definition 3.25).

The number of initial tokens of the CSW is set to

$$\begin{aligned} \mathrm{d}_0(b) &= \text{findTokenNumbers}(C,h) \\ &= \begin{cases} 1, & b \leqslant N_\mathrm{A} \\ r_a, & \mathrm{dst}(b) = (a,\imath), \quad a = i, \quad i \in I_\mathrm{F}, \quad \imath \in I_{\mathrm{U}_i} \\ 0, & otherwise \end{cases} \end{aligned} \tag{4.23}$$

where $r_a \in \mathbb{N}$ are the elements of the repetition vector (Definition 3.18) with the topology matrix (Definition 3.16) constructed using the rates in (3.132).

The method introduced in the definition above places a token in the SDFBs involved in self-loops and no tokens in the SDFBs with a rate converter port as the destination. The method places enough tokens in the SDFBs with a simulator as the destination so that execution of the CSW does not deadlock (Theorem 4.12). The next lemmas are used in the proof of Theorem 4.12.

Definition 4.9 (Labeled PASS search)**.** The labeled PASS search is Algorithm 3.2, which uses the following function to select SDFAs

$$\text{choose}\left(G, \mathrm{d}\left(2n\right), \mathbf{r}, n\right) = a', \quad \sum_{a=1}^{a'-1} r_a < n \leqslant \sum_{a=1}^{a'} m_a\left(n\right) \tag{4.24}$$

Lemma 4.10 (Simulator execution)**.** *Assume that the algorithm specified in Definition 4.9 is used to find a PASS. Let $a' \in I_\mathrm{F}$ and $n' = \sum_{a=1}^{a'-1} r_a$. Assume that the number of tokens before executing the simulator labeled a' is*

$$\mathrm{d}\left(2n', b\right) = \begin{cases} 1, & b \leqslant N_\mathrm{A} \\ 0, & \mathrm{dst}(b) = (a,\imath), \quad \imath \in I_{\mathscr{U}_a}, \quad a < a' \\ r_a, & \mathrm{src}(b) = (a,o), \quad o \in I_{\mathscr{Y}_a}, \quad a < a' \\ r_a, & \mathrm{dst}(b) = (a,\imath), \quad \imath \in I_{\mathscr{U}_a}, \quad a' \leqslant a \leqslant N_\mathrm{F} \\ 0, & \mathrm{src}(b) = (a,o), \quad o \in I_{\mathscr{Y}_a}, \quad a' \leqslant a \leqslant N_\mathrm{F} \end{cases} \tag{4.25}$$

The number of tokens in the SDFBs during execution of the simulator labeled $a' \leqslant N_\mathrm{F}$ is

$$\mathrm{d}(2n,b) = \begin{cases} 1, & b \leqslant N_\mathrm{A} \\ 0 & \mathrm{dst}(b) = (a,\imath), \quad \imath \in I_{\mathscr{U}_a}, \quad a < a' \\ r_a, & \mathrm{src}(b) = (a,o), \quad o \in I_{\mathscr{Y}_a}, \quad a < a' \\ r_a, & \mathrm{dst}(b) = (a,\imath), \quad \imath \in I_{\mathscr{U}_a}, \quad a' < a \leqslant N_\mathrm{F} \\ 0, & \mathrm{src}(b) = (a,o), \quad o \in I_{\mathscr{Y}_a}, \quad a' < a \leqslant N_\mathrm{F} \\ r_{a'} - n + n', & \mathrm{dst}(b) = (a',\imath), \quad \imath \in I_{\mathscr{U}_{a'}} \\ n - n', & \mathrm{src}(b) = (a',o), \quad o \in I_{\mathscr{Y}_{a'}} \end{cases} \tag{4.26}$$

for $n' < n \leqslant n' + r_{a'}$.

Proof. Let the label of the simulator $a' \leqslant N_\mathrm{F}$ be fixed. The proof is given by induction.

The induction basis is the case of $n = n' + 1$. The assumed number of tokens before the simulator is executed (4.25) fulfills the trigger condition (3.35) for the simulator. The number of tokens in the SDFBs after the simulator has consumed the tokens is determined by the equations (3.43) and (3.118)

$$\mathrm{d}\left(2n'+1\right) = \begin{cases} 1, & b \leqslant N_\mathrm{A} \\ 0 & \mathrm{dst}(b) = (a,\imath), \quad \imath \in I_{\mathscr{U}_a}, \quad a < a' \\ r_a, & \mathrm{src}(b) = (a,o), \quad o \in I_{\mathscr{Y}_a}, \quad a < a' \\ r_a, & \mathrm{dst}(b) = (a,\imath), \quad \imath \in I_{\mathscr{U}_a}, \quad a' < a \leqslant N_\mathrm{F} \\ 0, & \mathrm{src}(b) = (a,o), \quad o \in I_{\mathscr{Y}_a}, \quad a' \leqslant a \leqslant N_\mathrm{F} \\ r_{a'} - 1, & \mathrm{dst}(b) = (a',\imath), \quad \imath \in I_{\mathscr{U}_{a'}} \end{cases} \tag{4.27}$$

The number of tokens in the SDFBs after the simulator has produced the tokens is determined by the equations (3.43) and (3.119)

$$\mathrm{d}(2n) = \mathrm{d}\left(2n'+2\right) = \begin{cases} 1, & b \leqslant N_\mathrm{A} \\ 0 & \mathrm{dst}(b) = (a,\imath), \quad \imath \in I_{\mathscr{U}_a}, \quad a < a' \\ r_a, & \mathrm{src}(b) = (a,o), \quad o \in I_{\mathscr{Y}_a}, \quad a < a' \\ r_a, & \mathrm{dst}(b) = (a,\imath), \quad \imath \in I_{\mathscr{U}_a}, \quad a' < a \leqslant N_\mathrm{F} \\ 0, & \mathrm{src}(b) = (a,o), \quad o \in I_{\mathscr{Y}_a}, \quad a' < a \leqslant N_\mathrm{F} \\ r_{a'} - 1, & \mathrm{dst}(b) = (a',\imath), \quad \imath \in I_{\mathscr{U}_{a'}} \\ 1, & \mathrm{src}(b) = (a',o), \quad o \in I_{\mathscr{Y}_{a'}} \end{cases} \tag{4.28}$$

This proves the induction basis.

In the induction step it is assumed that (4.26) holds. It needs to be verified that the statement holds for the next invocation of the simulator $n' < n+1 \leqslant n' + r_{a'}$. The number of tokens in the SDFBs after the simulator has consumed the tokens is determined by the equations (3.43) and (3.118)

$$\mathrm{d}(2n+1) = \begin{cases} 1, & b \leqslant N_\mathrm{A} \\ 0 & \mathrm{dst}(b) = (a,\imath), \quad \imath \in I_{\mathscr{U}_a}, \quad a < a' \\ r_a, & \mathrm{src}(b) = (a,o), \quad o \in I_{\mathscr{Y}_a}, \quad a < a' \\ r_a, & \mathrm{dst}(b) = (a,\imath), \quad \imath \in I_{\mathscr{U}_a}, \quad a' < a \leqslant N_\mathrm{F} \\ 0, & \mathrm{src}(b) = (a,o), \quad o \in I_{\mathscr{Y}_a}, \quad a' < a \leqslant N_\mathrm{F} \\ r_{a'} - (n+1) + n', & \mathrm{dst}(b) = (a',\imath), \quad \imath \in I_{\mathscr{U}_{a'}} \\ n - n', & \mathrm{src}(b) = (a',o), \quad o \in I_{\mathscr{Y}_{a'}} \end{cases} \tag{4.29}$$

The number of tokens in the SDFBs after the simulator has produced the tokens is determined by the equations (3.43) and (3.119)

$$\mathrm{d}\big(2(n+1)\big) = \begin{cases} 1, & b \leqslant N_\mathrm{A} \\ 0 & \mathrm{dst}(b) = (a,\imath), \quad \imath \in I_{\mathscr{U}_a}, \quad a < a' \\ r_a, & \mathrm{src}(b) = (a,o), \quad o \in I_{\mathscr{Y}_a}, \quad a < a' \\ r_a, & \mathrm{dst}(b) = (a,\imath), \quad \imath \in I_{\mathscr{U}_a}, \quad a' < a \leqslant N_\mathrm{F} \\ 0, & \mathrm{src}(b) = (a,o), \quad o \in I_{\mathscr{Y}_a}, \quad a' < a \leqslant N_\mathrm{F} \\ r_{a'} - (n+1) + n', & \mathrm{dst}(b) = (a',\imath), \quad \imath \in I_{\mathscr{U}_{a'}} \\ (n+1) - n', & \mathrm{src}(b) = (a',o), \quad o \in I_{\mathscr{Y}_{a'}} \end{cases} \tag{4.30}$$

This proves the induction step. □

Lemma 4.11 (Rate converter execution)**.** *Assume that the algorithm specified in Definition 4.9 is used to find a PASS. Let $a' > N_\mathrm{F}$ and $n' = \sum_{a=1}^{a'-1} r_a$. Assume that the number of tokens before executing the rate converter labeled a' is*

$$\mathrm{d}(2n',b) = \begin{cases} 1, & b \leqslant N_\mathrm{A} \\ 0, & \mathrm{dst}(b) = (\underline{a},1), \quad \mathrm{src}(b) = (\hat{a},\hat{o}), \quad N_\mathrm{F} < \underline{a} < a' \\ r_{\check{a}}, & \mathrm{src}(b) = (\underline{a},1), \quad \mathrm{dst}(b) = (\check{a},\check{\imath}), \quad N_\mathrm{F} < \underline{a} < a' \\ r_{\check{a}}, & \mathrm{dst}(b) = (\underline{a},1), \quad \mathrm{src}(b) = (\hat{a},\hat{o}), \quad a' \leqslant \underline{a} \leqslant N_\mathrm{A} \\ 0, & \mathrm{src}(b) = (\underline{a},1), \quad \mathrm{dst}(b) = (\check{a},\check{\imath}), \quad a' \leqslant \underline{a} < N_\mathrm{A} \end{cases} \tag{4.31}$$

The number of tokens in the SDFBs during execution of the rate converter labeled $a' > N_\mathrm{F}$ is

$$\mathrm{d}(2n,b) = \begin{cases} 1, & b \leqslant N_\mathrm{A} \\ 0, & \mathrm{dst}(b) = (\underline{a},1), \quad \mathrm{src}(b) = (\hat{a},\hat{o}), \quad N_\mathrm{F} < \underline{a} < a' \\ r_{\check{a}}, & \mathrm{src}(b) = (\underline{a},1), \quad \mathrm{dst}(b) = (\check{a},\check{\imath}), \quad N_\mathrm{F} < \underline{a} < a' \\ r_{\check{a}}, & \mathrm{dst}(b) = (\underline{a},1), \quad \mathrm{src}(b) = (\hat{a},\hat{o}), \quad a' < \underline{a} \leqslant N_\mathrm{A} \\ 0, & \mathrm{src}(b) = (\underline{a},1), \quad \mathrm{dst}(b) = (\check{a},\check{\imath}), \quad a' < \underline{a} < N_\mathrm{A} \\ r_{\check{a}} - \mathrm{c}_{a'}(n-n'), & \mathrm{dst}(b) = (\underline{a},1), \quad \mathrm{src}(b) = (\hat{a},\hat{o}) \\ \mathrm{p}_{a'}(n-n'), & \mathrm{src}(b) = (\underline{a},1), \quad \mathrm{dst}(b) = (\check{a},\check{\imath}) \end{cases} \tag{4.32}$$

for $n' < n \leqslant n' + r_{a'}$.

Proof. Let the label of the rate converter $a' \in (N_\mathrm{F}, N_\mathrm{A}]$ be fixed. The proof is given by induction.

The induction basis is the case of $n = n' + 1$. The assumed number of tokens before the rate converter is executed (4.31) fulfills the trigger condition (3.35) for the rate converter. The number of tokens in the SDFBs after the rate converter has consumed the tokens is determined by the equations (3.43) and (3.127)

$$\mathrm{d}\left(2n'+1\right) = \begin{cases} 1, & b \leqslant N_\mathrm{A} \\ 0, & \mathrm{dst}(b) = (\underline{\mathrm{a}},1), \quad \mathrm{src}(b) = (\hat{a},\hat{o}), \quad N_\mathrm{F} < \underline{\mathrm{a}} < a' \\ r_{\check{a}}, & \mathrm{src}(b) = (\underline{\mathrm{a}},1), \quad \mathrm{dst}(b) = (\check{a},\check{\imath}), \quad N_\mathrm{F} < \underline{\mathrm{a}} < a' \\ r_{\check{a}}, & \mathrm{dst}(b) = (\underline{\mathrm{a}},1), \quad \mathrm{src}(b) = (\hat{a},\hat{o}), \quad a' < \underline{\mathrm{a}} \leqslant N_\mathrm{A} \\ 0, & \mathrm{src}(b) = (\underline{\mathrm{a}},1), \quad \mathrm{dst}(b) = (\check{a},\check{\imath}), \quad a' < \underline{\mathrm{a}} < N_\mathrm{A} \\ r_{\check{a}} - \mathrm{c}_{a'}, & \mathrm{dst}(b) = (\underline{\mathrm{a}},1), \quad \mathrm{src}(b) = (\hat{a},\hat{o}) \end{cases} \tag{4.33}$$

The number of tokens in the SDFBs after the rate converter has produced the tokens is determined by the equations (3.43) and (3.128)

$$\mathrm{d}(2n) = \mathrm{d}\left(2n'+2\right) = \begin{cases} 1, & b \leqslant N_\mathrm{A} \\ 0, & \mathrm{dst}(b) = (\underline{\mathrm{a}},1), \quad \mathrm{src}(b) = (\hat{a},\hat{o}), \quad N_\mathrm{F} < \underline{\mathrm{a}} < a' \\ r_{\check{a}}, & \mathrm{src}(b) = (\underline{\mathrm{a}},1), \quad \mathrm{dst}(b) = (\check{a},\check{\imath}), \quad N_\mathrm{F} < \underline{\mathrm{a}} < a' \\ r_{\check{a}}, & \mathrm{dst}(b) = (\underline{\mathrm{a}},1), \quad \mathrm{src}(b) = (\hat{a},\hat{o}), \quad a' < \underline{\mathrm{a}} \leqslant N_\mathrm{A} \\ 0, & \mathrm{src}(b) = (\underline{\mathrm{a}},1), \quad \mathrm{dst}(b) = (\check{a},\check{\imath}), \quad a' < \underline{\mathrm{a}} < N_\mathrm{A} \\ r_{\check{a}} - \mathrm{c}_{a'}, & \mathrm{dst}(b) = (\underline{\mathrm{a}},1), \quad \mathrm{src}(b) = (\hat{a},\hat{o}) \\ \mathrm{p}_{a'}, & \mathrm{src}(b) = (\underline{\mathrm{a}},1), \quad \mathrm{dst}(b) = (\check{a},\check{\imath}) \end{cases} \tag{4.34}$$

This proves the induction basis.

In the induction step it is assumed that (4.32) holds. It needs to be verified that the statement holds for the next invocation of the rate converter $n' < n+1 \leqslant n' + r_{a'}$. The number of tokens in the SDFBs after the rate converter has consumed the tokens is determined by the equations (3.43) and (3.127)

$$\mathrm{d}\left(2n+1\right) = \begin{cases} 1, & b \leqslant N_\mathrm{A} \\ 0, & \mathrm{dst}(b) = (\underline{\mathrm{a}},1)\,, \quad \mathrm{src}(b) = (\hat{a},\hat{o})\,, \quad N_\mathrm{F} < \underline{\mathrm{a}} < a' \\ r_{\breve{a}}, & \mathrm{src}(b) = (\underline{\mathrm{a}},1)\,, \quad \mathrm{dst}(b) = (\breve{a},\breve{\imath}), \quad N_\mathrm{F} < \underline{\mathrm{a}} < a' \\ r_{\breve{a}}, & \mathrm{dst}(b) = (\underline{\mathrm{a}},1)\,, \quad \mathrm{src}(b) = (\hat{a},\hat{o})\,, \quad a' < \underline{\mathrm{a}} \leqslant N_\mathrm{A} \\ 0, & \mathrm{src}(b) = (\underline{\mathrm{a}},1)\,, \quad \mathrm{dst}(b) = (\breve{a},\breve{\imath}), \quad a' < \underline{\mathrm{a}} < N_\mathrm{A} \\ r_{\breve{a}} - \mathrm{c}_{a'}\left(\left(n+1\right)-n'\right), & \mathrm{dst}(b) = (\underline{\mathrm{a}},1)\,, \quad \mathrm{src}(b) = (\hat{a},\hat{o}) \\ \mathrm{p}_{a'}\left(n-n'\right), & \mathrm{src}(b) = (\underline{\mathrm{a}},1)\,, \quad \mathrm{dst}(b) = (\breve{a},\breve{\imath}) \end{cases} \tag{4.35}$$

The number of tokens in the SDFBs after the rate converter has produced the tokens is determined by the equations (3.43) and (3.128)

$$\mathrm{d}\left(2\left(n+1\right)\right) = \begin{cases} 1, & b \leqslant N_\mathrm{A} \\ 0, & \mathrm{dst}(b) = (\underline{\mathrm{a}},1)\,, \quad \mathrm{src}(b) = (\hat{a},\hat{o})\,, \quad N_\mathrm{F} < \underline{\mathrm{a}} < a' \\ r_{\breve{a}}, & \mathrm{src}(b) = (\underline{\mathrm{a}},1)\,, \quad \mathrm{dst}(b) = (\breve{a},\breve{\imath}), \quad N_\mathrm{F} < \underline{\mathrm{a}} < a' \\ r_{\breve{a}}, & \mathrm{dst}(b) = (\underline{\mathrm{a}},1)\,, \quad \mathrm{src}(b) = (\hat{a},\hat{o})\,, \quad a' < \underline{\mathrm{a}} \leqslant N_\mathrm{A} \\ 0, & \mathrm{src}(b) = (\underline{\mathrm{a}},1)\,, \quad \mathrm{dst}(b) = (\breve{a},\breve{\imath}), \quad a' < \underline{\mathrm{a}} < N_\mathrm{A} \\ r_{\breve{a}} - \mathrm{c}_{a'}\left(\left(n+1\right)-n'\right), & \mathrm{dst}(b) = (\underline{\mathrm{a}},1)\,, \quad \mathrm{src}(b) = (\hat{a},\hat{o}) \\ \mathrm{p}_{a'}\left(\left(n+1\right)-n'\right), & \mathrm{src}(b) = (\underline{\mathrm{a}},1)\,, \quad \mathrm{dst}(b) = (\breve{a},\breve{\imath}) \end{cases} \tag{4.36}$$

This proves the induction step. □

Theorem 4.12 (Sequential execution)**.** *Assume that the number of initial tokens* d_0 *for a CSW (Definition 3.25) is determined using the method described in Definition 4.8. Assume that the algorithm specified in Definition 4.9 is used to find a PASS. Let* $n'' = \sum_{a=1}^{a''} r_a$*. The number of* *tokens in the SDFBs is*

$$\mathrm{d}\left(2n'',b\right) = \begin{cases} 1, & b \leqslant N_\mathrm{A} \\ 0, & \mathrm{dst}(b) = (a,\iota), \quad \iota \in I_{\mathscr{U}_a}, \quad a \leqslant a'' \\ r_a, & \mathrm{src}(b) = (a,o), \quad o \in I_{\mathscr{Y}_a}, \quad a \leqslant a'' \\ r_a, & \mathrm{dst}(b) = (a,\iota), \quad \iota \in I_{\mathscr{U}_a}, \quad a'' < a \leqslant N_\mathrm{F} \\ 0, & \mathrm{src}(b) = (a,o), \quad o \in I_{\mathscr{Y}_a}, \quad a'' < a \leqslant N_\mathrm{F} \end{cases} \tag{4.37}$$

for simulators $a'' \leqslant N_\mathrm{F}$ and

$$\mathrm{d}\left(2n'',b\right) = \begin{cases} 1, & b \leqslant N_\mathrm{A} \\ 0, & \mathrm{dst}(b) = (\underline{a},1), \quad \mathrm{src}(b) = (\hat{a},\hat{o}), \quad N_\mathrm{F} < \underline{a} \leqslant a'' \\ r_{\breve{a}}, & \mathrm{src}(b) = (\underline{a},1), \quad \mathrm{dst}(b) = (\breve{a},\breve{\iota}), \quad N_\mathrm{F} < \underline{a} \leqslant a'' \\ r_{\breve{a}}, & \mathrm{dst}(b) = (\underline{a},1), \quad \mathrm{src}(b) = (\hat{a},\hat{o}), \quad a'' < \underline{a} \leqslant N_\mathrm{A} \\ 0, & \mathrm{src}(b) = (\underline{a},1), \quad \mathrm{dst}(b) = (\breve{a},\breve{\iota}), \quad a'' < \underline{a} < N_\mathrm{A} \end{cases} \tag{4.38}$$

for rate converters $a'' > N_\mathrm{F}$.

Proof. First, the statement for simulators $a'' \leqslant N_\mathrm{F}$ is proven by induction. The induction basis is the case of $a'' = 1'$. The execution step index is then $n'' = r_1$. The number of initial tokens d_0, which is determined by the method described in definition 4.8, fulfills the condition (4.25). Lemma 4.10 proves the induction basis

$$\mathrm{d}\left(2n'',b\right) = \begin{cases} 1, & b \leqslant N_\mathrm{A} \\ 0, & \mathrm{dst}(b) = (a,\iota), \quad \iota \in I_{\mathscr{U}_a}, \quad a = a'' = 1 \\ r_1, & \mathrm{src}(b) = (a,o), \quad o \in I_{\mathscr{Y}_a}, \quad a = a'' = 1 \\ r_a, & \mathrm{dst}(b) = (a,\iota), \quad \iota \in I_{\mathscr{U}_a}, \quad a'' < a \leqslant N_\mathrm{F} \\ 0, & \mathrm{src}(b) = (a,o), \quad o \in I_{\mathscr{Y}_a}, \quad a'' + 1 < a < N_\mathrm{F} \end{cases} \tag{4.39}$$

In the induction step it is assumed that (4.37) holds. The observed execution step has the index

$n'' + r_{a''+1}$. Lemma 4.10 is used to prove the induction step

$$\mathrm{d}\left(2\left(n''+r_{a''+1}\right),b\right) = \begin{cases} 1, & b \leqslant N_\mathrm{A} \\ 0, & \mathrm{dst}(b) = (a,\imath), \quad \imath \in I_{\mathscr{U}_a}, \quad a \leqslant a''+1 \\ r_a, & \mathrm{src}(b) = (a,o), \quad o \in I_{\mathscr{Y}_a}, \quad a \leqslant a''+1 \\ r_a, & \mathrm{dst}(b) = (a,\imath), \quad \imath \in I_{\mathscr{U}_a}, \quad a''+1 < a \leqslant N_\mathrm{F} \\ 0, & \mathrm{src}(b) = (a,o), \quad o \in I_{\mathscr{Y}_a}, \quad a''+1 < a < N_\mathrm{F} \end{cases} \tag{4.40}$$

and hence the statement of the theorem for simulators.

In the second part of the proof, the statement for rate converters $a'' > N_\mathrm{F}$ is proven by induction. The induction basis is the case of $a'' = N_\mathrm{F}+1$. The execution step index is then $n'' = \sum_{a=1}^{N_\mathrm{F}+1} r_a$. The first part of the proof shows that the number of tokens $\mathrm{d}(n') = \mathrm{d}\left(\sum_{a=1}^{N_\mathrm{F}+1} r_a\right)$ fulfills the condition (4.31). Equations (4.31) and (4.37) describe the same number of tokens for the execution step with index n'. This statement follows from (3.134), (3.135), $a' = N_\mathrm{F}$ and $a'' = N_\mathrm{F}+1$. The rate converter connections are used to designate the buffers in (4.31) and simulator connections in (4.25). Lemma 4.11 proves the induction basis

$$\mathrm{d}\left(2n'',b\right) = \begin{cases} 1, & b \leqslant N_\mathrm{A} \\ 0, & \mathrm{dst}(b) = (\underline{a},1), \quad \mathrm{src}(b) = (\hat{a},\hat{o}), \quad \underline{a} = a'' = N_\mathrm{F}+1 \\ r_{\breve{a}}, & \mathrm{src}(b) = (\underline{a},1), \quad \mathrm{dst}(b) = (\breve{a},\breve{\imath}), \quad \underline{a} = a'' = N_\mathrm{F}+1 \\ r_{\breve{a}}, & \mathrm{dst}(b) = (\underline{a},1), \quad \mathrm{src}(b) = (\hat{a},\hat{o}), \quad a'' < \underline{a} \leqslant N_\mathrm{A} \\ 0, & \mathrm{src}(b) = (\underline{a},1), \quad \mathrm{dst}(b) = (\breve{a},\breve{\imath}), \quad a'' < \underline{a} < N_\mathrm{A} \end{cases} \tag{4.41}$$

In the induction step it is assumed that (4.38) holds. The observed execution step has the index $n'' + r_{a''+1}$. Lemma 4.11 is used to prove the induction step

$$\mathrm{d}\left(2\left(n''+r_{a''+1}\right),b\right) = \begin{cases} 1, & b \leqslant N_\mathrm{A} \\ 0, & \mathrm{dst}(b) = (\underline{a},1), \quad \mathrm{src}(b) = (\hat{a},\hat{o}), \quad N_\mathrm{F} < \underline{a} \leqslant a''+1 \\ r_{\breve{a}}, & \mathrm{src}(b) = (\underline{a},1), \quad \mathrm{dst}(b) = (\breve{a},\breve{\imath}), \quad N_\mathrm{F} < \underline{a} \leqslant a''+1 \\ r_{\breve{a}}, & \mathrm{dst}(b) = (\underline{a},1), \quad \mathrm{src}(b) = (\hat{a},\hat{o}), \quad a''+1 < \underline{a} \leqslant N_\mathrm{A} \\ 0, & \mathrm{src}(b) = (\underline{a},1), \quad \mathrm{dst}(b) = (\breve{a},\breve{\imath}), \quad a''+1 < \underline{a} < N_\mathrm{A} \end{cases} \tag{4.42}$$

and hence the statement of the theorem for rate converters. □

Corollary 4.13 (Nonterminating execution)**.** *Assume that the initial tokens* d_0 *for a CSW (Definition 3.25) are obtained using the method described in Definition 4.8. Such an SDFG does not deadlock.*

Proof. This corollary is a direct consequence of Theorem 3.10 and Theorem 4.12. □

Corollary 4.13 states that the method described in Definition 4.8 ensures that the CSW (Definition 3.25) does not deadlock. The rest of the section shows that such a CSW can be executed in real time (Theorem 4.17).

Assumption 4.14 (Execution time)**.** Suppose the co-simulation network wrapper (Definition 3.25) is running on the IPEP (Definition 4.1). The execution time for simulators $a \leqslant N_{\mathrm{F}}$ is

$$\mathrm{time}(a) \leqslant \mathrm{h}(a) \tag{4.43}$$

and the execution time for rate converters $a > N_{\mathrm{F}}$ is

$$\mathrm{time}(a) = 0 \tag{4.44}$$

The previous assumption states that the time behavior of a simulator $a \leqslant N_{\mathrm{F}}$ corresponds to the time behavior of hardware. In hardware components there is no concept of simulated time. The communication interface to the hardware component can be modeled with a simulator (Definition 3.23). It is assumed that the rate converters (Definition 3.24) run in zero time. The rate conversion (3.129) is a simple calculation and should be negligible in practice compared to simulator calculations.

Lemma 4.15 (Simulator completion time)**.** *Assume that the method described in Definition 4.8 is used to set the number of initial CSW tokens (Definition 3.25) and Assumption 4.14 holds. The completion time of simulator $a \leqslant N_{\mathrm{F}}$ invocations is*

$$T(G, a, k_a) \leqslant k_a\, \mathrm{h}(a), \quad (a, k_a) \in \mathit{iteration}(G) \tag{4.45}$$

Proof. Let $a \leqslant N_{\mathrm{F}}$ be fixed. The proof is given by induction.

The induction basis is the case of $k_a = 1$. With the number of tokens specified in Definition 4.8, there are no precedence constraints (4.9) that are met regardless of the number executions of the connected SDFAs. This statement is verified by substitution

$$k_{\breve{a}} = k_a = 1, \quad \breve{\imath} = \imath, \quad \mathrm{c}_{\breve{a}}(\breve{\imath}) = \mathrm{c}_a(\imath) = 1 \tag{4.46}$$

to (4.9). The number of initial tokens in the input SDFBs is

$$\mathrm{d}_0(b) = \begin{cases} 1, & (a, \imath) = \mathrm{dst}(b), \quad \imath = N_{\mathscr{U}_a} \\ r_a, & (a, \imath) = \mathrm{dst}(b), \quad \imath \in I_{\mathrm{U}_a} \end{cases} \tag{4.47}$$

The inequality (4.9) does not hold for any input port of the simulator since

$$\left(k_{\underline{a}}-1\right)\mathrm{p}_{\underline{a}}\left(\underline{o}\right)+\mathrm{d}_0(b)\geqslant \mathrm{d}_0(b)\geqslant k_a\mathrm{c}_a(\iota)=1 \tag{4.48}$$

The consumption rate in the substitution is set by the definition of the simulator (3.118). Therefore precedences$(G,a,1)=\emptyset$. The induction basis is proven by (4.16) and (4.43), i.e.

$$T(G,a,1)=\mathrm{time}\,(a)\leqslant \mathrm{h}\,(a) \tag{4.49}$$

In the induction step it is assumed that (4.45) holds. It needs to be verified that the statement holds for the next invocation of the simulator. The invocation of the simulator depends on the previous invocations of the same simulator

$$\mathrm{precedences}(G,a,k_a+1)=\left\{(a,k_a')\ :\ k_a'\leqslant k_a\right\} \tag{4.50}$$

This statement is verified by substitution

$$k_{\breve{a}}=k_a+1\leqslant r_a,\quad \breve{\iota}=\iota,\quad \mathrm{c}_{\breve{a}}\left(\breve{\iota}\right)=\mathrm{c}_a(\iota)=1 \tag{4.51}$$

to (4.9). The inequality (4.9) does not hold for input ports of the simulator $\iota\in I_{\mathrm{U}_a}$

$$\left(k_{\underline{a}}-1\right)\mathrm{p}_{a'}(\underline{o})+\mathrm{d}_0(b)\geqslant \mathrm{d}_0(b)=r_a>(k_a-1)\,\mathrm{c}_{a''}\left(\iota''\right)=k_a \tag{4.52}$$

The inequality (4.9) holds for the self-loop of the simulator (3.5), (3.6) and (3.8)

$$(k_a+1)\ \mathrm{c}_a(N_{\mathscr{U}_a})=k_a+1>(k-1)\ \mathrm{p}_a(N_{\mathscr{Y}_a})+\mathrm{d}_0(b)=k,\quad k\leqslant k_a \tag{4.53}$$

The induction step is proven by (4.16), (4.43) and (4.50). The completion time of the next invocation of a simulator is

$$T(G,a,k_a+1)=\mathrm{time}\big(\mathrm{A}(a)\big)+T(G,a,k_a)\leqslant (k_a+1)\ \mathrm{h}(a) \tag{4.54}$$

□

Lemma 4.16 (Rate converter completion time)**.** *Assume that the method described in Definition 4.8 is used and Assumption 4.14 holds. The completion time of rate converter* $\underline{a}>N_{\mathrm{F}}$ *invocations is*

$$T(G,\underline{a},k_{\underline{a}})\leqslant H,\quad (\underline{a},k_{\underline{a}})\in \mathit{iteration}(G) \tag{4.55}$$

Proof. Let $\underline{a}>N_{\mathrm{F}}$ be fixed. The proof is given by induction.

The induction basis is the case of $k_{\underline{a}}=1$. Definition 4.8 specifies the number of initial tokens in each SDFB connected to a rate converter's input port $(\underline{\mathrm{a}}, \imath) = \mathrm{dst}(b)$. The set of precedence constraints for the first invocation of the rate converter is

$$\mathrm{precedences}(G, \underline{\mathrm{a}}, 1) = \left\{(\hat{a}, k_{\hat{a}}) \;:\; k_{\hat{a}} \in \mathbb{N}, \quad k_{\hat{a}} \leqslant \mathrm{c}_{\underline{\mathrm{a}}}(1)\right\} \tag{4.56}$$

where the simulator index $\hat{a}$ and simulator's output port index $\hat{o}$ are obtained by (3.141). The preconditions are verified by substitution

$$\breve{a} = \underline{\mathrm{a}}, \quad k_{\breve{a}} = k_{\underline{\mathrm{a}}}, \breve{\imath} = 1, \quad \mathrm{p}_{\hat{a}}(\hat{o}) = 1, \quad \mathrm{c}_{\breve{a}}(\breve{\imath}) = \mathrm{c}_{\underline{\mathrm{a}}}(1) \tag{4.57}$$

to (4.9) where the consumption rate of the rate converter is specified by (3.132). The inequality (4.9) holds for $k_{\hat{a}} \leqslant \mathrm{c}_{\underline{\mathrm{a}}}(1)$ and $k_{\breve{a}} = 1$

$$(k_{\hat{a}} - 1)\mathrm{p}_{\hat{a}}(\hat{o}) + \mathrm{d}_0(b) = k_{\hat{a}} - 1 < k_{\breve{a}}\mathrm{c}_{\underline{\mathrm{a}}}(1) = \mathrm{c}_{\underline{\mathrm{a}}}(1) \tag{4.58}$$

The self-loop does not block the first invocation of the rate converter. This statement is verified by substitution

$$\begin{gathered}\hat{a} = \underline{\mathrm{a}}, \quad \breve{a} = \underline{\mathrm{a}}, \quad k_{\hat{a}} = k_{\underline{\mathrm{a}}} - 1 = 1, \quad k_{\breve{a}} = k_{\underline{\mathrm{a}}} = 1, \quad \hat{o} = 2, \\ \breve{\imath} = 2, \quad \mathrm{p}_{\hat{a}}(\hat{o}) = \mathrm{p}_{\underline{\mathrm{a}}}(2) = 1, \quad \mathrm{c}_{\breve{a}}(\breve{\imath}) = \mathrm{c}_{\underline{\mathrm{a}}}(2) = 1\end{gathered} \tag{4.59}$$

to (4.9). The SDFB on the self-loop (3.140) has single initial token (4.23)

$$\mathrm{d}_0(b) = 1, \quad (\underline{\mathrm{a}}, 2) = \mathrm{src}(b), \quad (\underline{\mathrm{a}}, 2) = \mathrm{dst}(b) \tag{4.60}$$

The inequality (4.9) does not hold for the self-loop of the rate converter since

$$(k_{\hat{a}} - 1)\,\mathrm{p}_{\hat{a}}(\hat{o}) + \mathrm{d}_0(b) = 1 = k_{\breve{a}}\mathrm{c}_{\breve{a}}(\breve{\imath}) \tag{4.61}$$

The induction basis is proven by Lemma 4.15, (4.16) and (4.44). Each invocation of the simulator has a completion time $T(G, \hat{a}, k_{\hat{a}}) \leqslant H$. Since it is assumed that rate converters are executed instantaneously (4.44) the induction basis is proven

$$T(G, \underline{\mathrm{a}}, 1) = \max_{(\hat{a}, k_{\hat{a}}) \in \mathrm{precedences}(G, \underline{\mathrm{a}}, 1)} T(G, \hat{a}, k_{\hat{a}}) \leqslant H \tag{4.62}$$

In the induction step it is assumed that (4.55) holds. It needs to be verified that the statement holds for the next invocation of the rate. The set of precedence constraints for the next invocation of the rate converter is

$$\mathrm{precedences}(G, \underline{\mathrm{a}}, k_{\underline{\mathrm{a}}} + 1) = \left\{(\underline{\mathrm{a}}, k'_{\underline{\mathrm{a}}}) \;:\; k'_{\underline{\mathrm{a}}} \leqslant k_{\underline{\mathrm{a}}}\right\} \cup \left\{(\hat{a}, k_{\hat{a}}) \;:\; k_{\hat{a}} \leqslant \left(k_{\underline{\mathrm{a}}} + 1\right)\mathrm{c}_{\underline{\mathrm{a}}}(1)\right\} \tag{4.63}$$

where the simulator index $\hat{a}$ and simulator's output port index $\hat{o}$ are obtained by (3.141). The preconditions for the execution of the simulator are verified by substitution (4.57) to (4.9) where the consumption rate of the rate converter is specified by (3.132). The inequality (4.9) holds for $k_{\hat{a}} \leqslant (k_{\underline{a}} + 1)\ c_{\underline{a}}(1)$

$$(k_{\hat{a}} - 1)\mathrm{p}_{\hat{a}}(\hat{o}) = k_{\hat{a}} - 1 < k_{\check{a}}\mathrm{c}_{\check{a}}(\check{\imath}) = (k_{\underline{a}} + 1)\mathrm{c}_{\underline{a}}(1) \tag{4.64}$$

The precondition for the previous execution of the rate converter is verified by substitution (4.59) to (4.9). The SDFB on the self-loop has single initial token (4.60). The inequality (4.9) holds for the self-loop of the rate converter $k'_{\underline{a}} \leqslant k_{\underline{a}}$

$$(k_{\hat{a}} - 1)\mathrm{p}_{\hat{a}}(\hat{o}) + \mathrm{d}_0(b) = k'_{\underline{a}} \leqslant k_{\underline{a}} < k_{\check{a}}\mathrm{c}_{\check{a}}(\check{\imath}) = k_{\underline{a}} + 1 \tag{4.65}$$

The induction step is proven by Lemma 4.15, (4.16) and (4.44). Lemma 4.15 implies that invocation of the simulator has a completion time $T(G, \hat{a}, k_{\hat{a}}) \leqslant H$. The induction step assumes that the previous invocation of the rate converter has a completion time $T(G, \underline{a}, k) \leqslant H$. Since it is assumed that rate converters are executed instantaneously (4.44) the induction step is proven

$$T(G, \underline{a}, k_{\underline{a}} + 1) = \max_{(a,k_a) \in \mathrm{precedences}(G,\underline{a},k_{\underline{a}}+1)} T(G, a, k_a) \leqslant H \tag{4.66}$$

□

Theorem 4.17 (Execution time $\leqslant$ simulated time)**.** *Assume that the method described in Definition 4.8 is used and Assumption 4.14 holds. The makespan (Definition 4.6) of the CSW (Definition 3.25) is less than the simulated time increment (Theorem 3.29)*

$$T_{iteration}(G) \leqslant H \tag{4.67}$$

Proof. Lemma 4.15 and Theorem 3.29 show that for simulators $a \leqslant N_{\mathrm{F}}$, the completion time of all the invocations in the iteration is bounded by the simulated time increment

$$T(G, a, k) \leqslant k_a\, \mathrm{h}(a) \leqslant H = r_a\, \mathrm{h}(a), \quad a \leqslant N_{\mathrm{F}}, \quad k_a \leqslant r_a \tag{4.68}$$

Lemma 4.16 proves that for rate converters $a > N_{\mathrm{F}}$, the completion time of all the invocations in the iteration is bounded by the simulated time increment. The statement of the theorem follows from the definition of makespan (Definition 4.6). □

Theorem 4.17 shows that the CSW can be executed in real time with the number of initial tokens calculated using the method in Definition 4.8. The simulated time increment for such a CSW is shorter than the execution time of a single SDFI.

4.3 Model based development

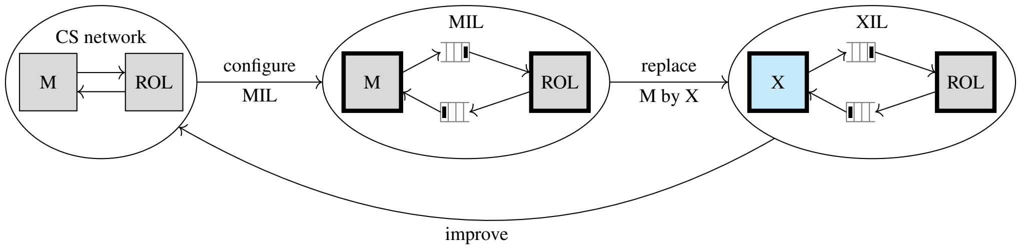


Figure 4.1: This thesis assumes that each phase of model-based development (MBD) runs an X in the loop (XIL) simulation and that the first phase always runs a model in the loop (MIL) simulation. The information obtained through a later phase of MBD can be used to improve the design in earlier phases.

Section 4.1 introduces the analysis of execution time on the IPEP (Definition 4.1). Section 4.2 shows how to calculate initial tokens of a CSW that enable real-time execution (Definition 4.8). This section tries to explain the motivation and benefits of modeling the execution of a CSN (Definition 2.14) by a CSW (Definition 3.25). Model-based development (MBD) is considered the main motivation for the research carried out in this thesis. This section motivates the MBD (Figure 4.1), in which the phases are XIL simulations and MIL simulation is the first phase. An X in the loop (XIL) simulation is a co-simulation experiment (Definition 4.18). Such a development is a cyclical process in which the results of later phases are compared with earlier phases. If a single SDFA is replaced between two MBD phases, only this change is responsible for the different results. Example 4.19 and 4.20 are used to demonstrate the MBD of a simple control loop.

Definition 4.18 (Co-simulation experiment)**.** A co-simulation experiment is a tuple

$$E = (G, t_{end}) \tag{4.69}$$

where

- $G \in \mathcal{G}$ is an SDFG (Definition 3.2),
- and $t_{end} \in \mathbb{Q}_{>0}$ is the duration of the experiment.

Let

$$G' = \text{networkToDataflow}(C, h, \mathrm{d}_0, \omega_0) \tag{4.70}$$

be a CSW (Definition 3.25) with the same behavior as G, i.e.

$$\omega_{\text{total}}(b, \theta_{total}) = \omega'_{\text{total}}(b, \theta_{total}) \tag{4.71}$$

where

- $\omega_{\text{total}} : I^{\mathrm{B}} \times \mathbb{N}_0 \to \mathbb{N}_0$ is the sequence of tokens visiting the SDFBs of G,

•and $\omega'_{\text{total}} : I^{\text{B}} \times \mathbb{N}_0 \to \mathbb{N}_0$ is the sequence of tokens visiting the SDFBs of G'.
The results of the co-simulation experiment E can be obtained by simulating G' using Algorithm 3.1 until the simulated time of all slaves does not exceed t_{end}.

In the above definition of the co-simulation experiment, the SDFG (Definition 3.2) does not need to be a CSW (Definition 3.25). The reason is that a CSW is constructed from a CSN (Definition 2.14), i.e. all of the simulators are constructed from an FMU (Definition 2.13). Within the MBD shown in Figure 4.1 SDFAs can be implemented as software or hardware components. However, a CSW is assumed to describe the SDFG behavior. This makes it possible to replace a simulator with other hardware or software. These components do not have to correspond to the FMU interface (Definition 2.13). In this thesis a XIL simulation is used as an alias for the co-simulation experiment. The hardware or software that replaces the simulator marks the X in the XIL simulation. When hardware replaces a simulator, a co-simulation experiment is a hardware in the loop (HIL) simulation. When software replaces a simulator, a co-simulation experiment is software in the loop (SIL) simulation. A model in the loop (MIL) is a co-simulation experiment in which the SDFG is a CSW. If the same CSW is used to describe the two phases of MBD, their results can be compared and the models can be improved based on the comparison. Example 4.19 introduces the MIL simulation of a simple control loop. Example 4.20 shows how a mistake in the programming can be detected using the SIL simulation.

Example 4.19 (Control loop MIL simulation)**.** Assume that the control system that is modeled with the CODESUB presented in Example 2.6. Assume that this CODESUB is wrapped with the SYSW presented in Example 2.18. Let C denote the CODESUB (2.15) and C the CSN (2.89). Assume that the control system will run the controller algorithm with the frequency of 1000 Hz. Assume that the controlled process signals are sampled with 500 Hz. This allows to determine the step-sizes of the simulators

$$\mathrm{h}(i) = \begin{cases} \frac{1}{1000}, & i = 1 \\ \frac{1}{500}, & i = 2 \end{cases} \tag{4.72}$$

The number of initial tokens of the CSW is determined by the method presented in Definition 4.8

$$\mathrm{d}_0(b) = \begin{cases} 1, & b \in \{1,2,4\} \\ 2, & b = 3 \\ 0, & otherwise \end{cases} \tag{4.73}$$

The values of initial tokens of the CSW are

$$\omega_0(b,\theta) = \begin{cases} v_{01}, & b = 1, \quad \theta = 1 \\ v_{02}, & b = 2, \quad \theta = 1 \\ 0, & b = 3, \quad \theta \in \{1,2\} \\ 0, & b = 4, \quad \theta = 1 \end{cases} \tag{4.74}$$

The duration of the co-simulation experiment is equal to

$$t_{end} = 20 \tag{4.75}$$

The MIL simulation of the control loop is the co-simulation experiment

$$E^{MIL} = (G^{MIL}, t_{end}) \tag{4.76}$$

where

$$G^{MIL} = \text{networkToDataflow}(C, h, \mathrm{d}_0, \omega_0) \tag{4.77}$$

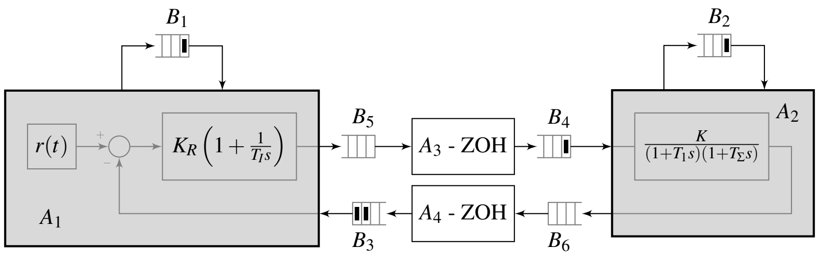
The construction of this CSW is implemented in Python procedure `sdf4sim.example.control.mil_mbd` [44
The result of the MIL simulation is available in Figure 4.4.

The CSW presented in the previous example models a multi-rate execution (Figure 4.2a), while the one in Example 3.30 models a single-rate execution (Figure 3.3a). The one in (4.77) presents a multi-rate execution. Example 3.30 and Example 4.19 are very similar. The only difference is that in Example 4.19 the information about the assumed HIL used to create the CSW. Such a CSW takes into account the communication frequencies of the hardware components. This makes the CSW in Example 4.19 a more accurate model of the developed technical system. The following example continues the MBD. It shows how a SIL simulation can be used to develop and debug controller code.

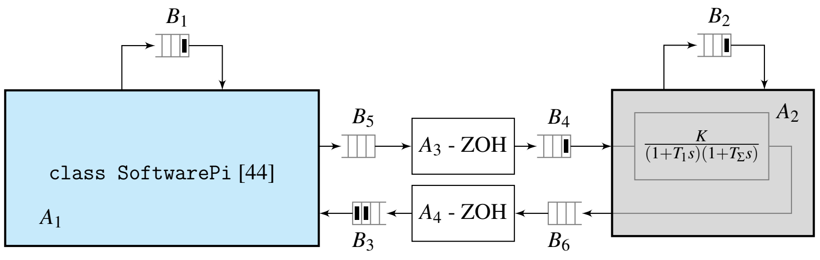
Example 4.20 (Control loop SIL simulation)**.** In this example, a software component is developed with the help of the process described in Figure 4.3. The SIL simulation of the control system should match the behavior of the MIL simulation shown in Example 4.19. The developed component is the controller algorithm and its implementation is repeated until the previous specification is met. The SIL simulation is obtained by replacing the controller simulator in the CSW (4.77). Assume that the controller has a single parameter different compared to the one shown in Example 2.6

$$T_I = 5, \quad K_R = 2 \tag{4.78}$$

and that SDFA ${A_1}^{SIL_1}$ is a Python implementation of such a controller. Let G^{MIL} be the SDFG used to execute the MIL simulation (4.76) and A^{MIL} be the function that labels its SDFAs. Then


B_1
B_2
B_5
B_4
$r(t)$
$K_R\left(1+\frac{1}{T_I s}\right)$
A_3 - ZOH
A_4 - ZOH
$\frac{K}{(1+T_1 s)(1+T_\Sigma s)}$
A_2
A_1
B_3
B_6

(a) MIL simulation



(b) SIL simulation

Figure 4.2: Example 4.20

SIL simulation

$$E^{SIL_1} = (G^{SIL_1}, t_{end}) \tag{4.79}$$

where the SDFAs are

$$\mathrm{A}^{SIL_1}(i) = \begin{cases} A_1{}^{SIL_1}, & i = 1 \\ \mathrm{A}^{MIL}, & otherwise \end{cases} \tag{4.80}$$

In Figure 4.4 it can be seen that the control system with such an implementation of the controller does not match the behavior of the MIL simulation (4.76). After the implementation of the controller is changed to have the correct parameters (2.24) and the following SIL simulation is executed

$$E^{SIL_2} = (E^{SIL_2}, t_{end}) \tag{4.81}$$

This controller ensures that the SIL simulation behavior matches the behavior of the MIL simulation (Figure 4.4). The code used to execute MIL, SIL_1 and SIL_2 can be seen in Python

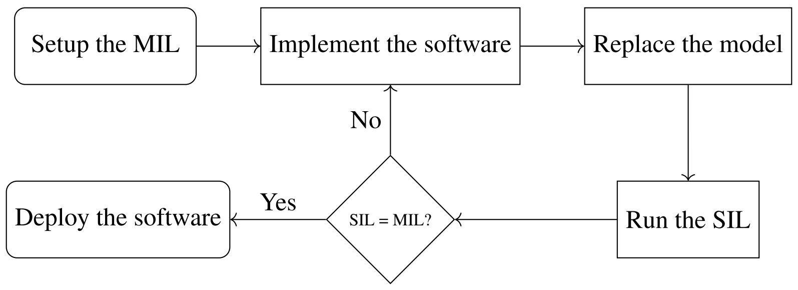


Figure 4.3: If the software is being implemented using MBD procedure suggested in Figure 4.1 the above flow chart can be followed to make sure it is properly implemented and tested.

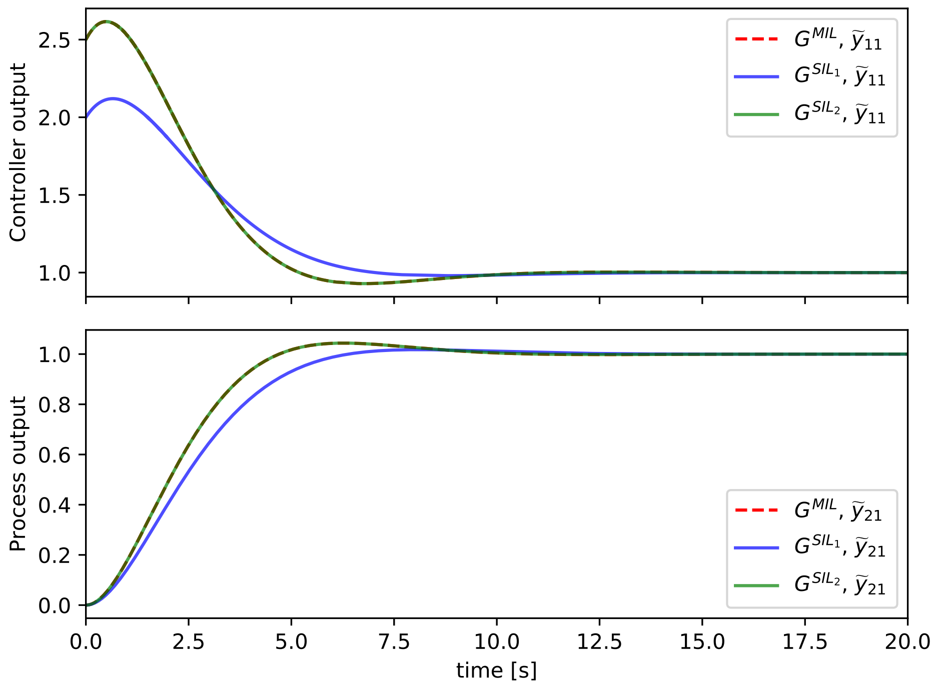


Figure 4.4: The MIL simulation presented in Example 4.19 describes the desired behavior of the controlled system. The SIL simulation is executed once with a implementation that does not follow this behavior. After the correction (Figure 4.3) the behavior of the second SIL implementation matches the behavior of the MIL simulation.

procedure `sdf4sim.example.control.mbd_comparison` at [44].

In the previous example, two SIL simulations (4.79) and (4.81) are executed and compared with the MIL simulation (4.76). These two executions are iterations of the process shown in Figure 4.3. Such a process can be viewed as a test-driven development process [75]. From this point of view, the test assertion is that the SIL simulation performed with the correctly implemented software component delivers the same results as the reference MIL simulation.

In Example 4.19, step sizes of simulators that correspond to the HIL simulation setup (4.72) are selected. The HIL simulation setup is left for future work. The authors in [28] state that "SDF is useful as hardware description, but our intent is that SDF be used primarily for functional description". Since SDF is mainly used to verify embedded systems, setting up the HIL simulation should be a simple and time consuming task.

The examples presented in this section serve to motivate the use of SDFG as the basis for MBD (Figure 4.1). MBD has many variations and this section covers only a simple variation using examples. Rigorous analysis of MBD will be a topic of future work.

Chapter 5

Defect calculation

Defect control is a well-researched technique for controlling numerical errors [47, 61]. The defect control for systems of differential and algebraic equations was investigated in [41]. An advantage of defect control is that it enables the distribution of responsibility for numerical errors in the co-simulation (Figure 5.1). This is one of the main contributions of this thesis, as existing co-simulation error analysis does not explicitly consider the distribution of responsibility for numerical errors [39, 40]. This thesis implies that the responsibility for the integration defect is assigned to the solvers and the connection defect to the CSM. This is reflected in Figure 5.1, in which connection equations are colored white because they are known to the CSM. The internal state equations of the CSLs are colored dark grey because they are not available to the CSM. The output equations are colored light gray because the output defect is estimated and controlled by the CSM in this chapter.

Theorem 2.17 and Theorem 5.4 show that connection and output defects can be controlled by reducing the simulator step sizes. These theorems also suggest what simulator step sizes can be reduced to ensure small target connection and output defects. Section 5.1 describes the calculation of the connection defect for the CSW. Section 5.2 shows how the output defect can

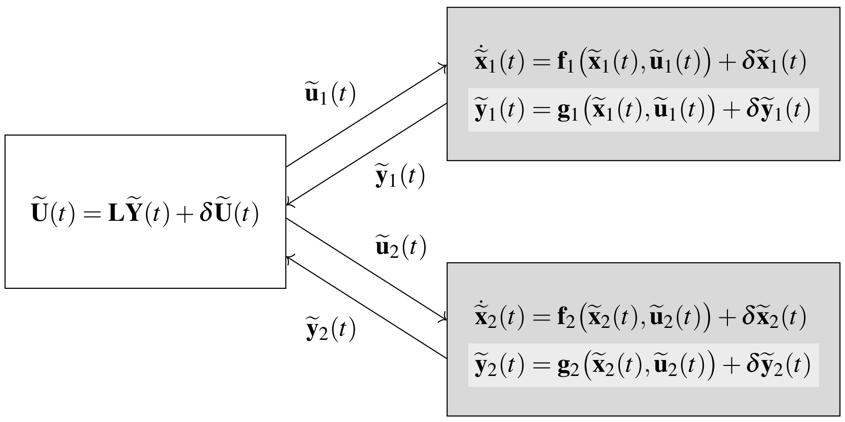

$\dot{\widetilde{\mathbf{x}}}_1(t) = \mathbf{f}_1\big(\widetilde{\mathbf{x}}_1(t), \widetilde{\mathbf{u}}_1(t)\big) + \delta\widetilde{\mathbf{x}}_1(t)$
$\widetilde{\mathbf{y}}_1(t) = \mathbf{g}_1\big(\widetilde{\mathbf{x}}_1(t), \widetilde{\mathbf{u}}_1(t)\big) + \delta\widetilde{\mathbf{y}}_1(t)$
$\widetilde{\mathbf{u}}_1(t)$
$\widetilde{\mathbf{y}}_1(t)$
$\widetilde{\mathbf{U}}(t) = \mathbf{L}\widetilde{\mathbf{Y}}(t) + \delta\widetilde{\mathbf{U}}(t)$
$\widetilde{\mathbf{u}}_2(t)$
$\widetilde{\mathbf{y}}_2(t)$
$\dot{\widetilde{\mathbf{x}}}_2(t) = \mathbf{f}_2\big(\widetilde{\mathbf{x}}_2(t), \widetilde{\mathbf{u}}_2(t)\big) + \delta\widetilde{\mathbf{x}}_2(t)$
$\widetilde{\mathbf{y}}_2(t) = \mathbf{g}_2\big(\widetilde{\mathbf{x}}_2(t), \widetilde{\mathbf{u}}_2(t)\big) + \delta\widetilde{\mathbf{y}}_2(t)$

Figure 5.1: CSLs can monitor integration defects (dark grey). The CSM can calculate the connection defect (white) and estimate the output defect (light grey).

be estimated with the extension of the CSW. Theorem 2.12 shows that the co-simulation error is limited when the co-simulation defect is limited.

The topic of Section 5.3 is the comparison of CSWs. The proposed defect calculation can be used to define a criterion for this purpose (Definition 5.10). Such a criterion can be used to find a suitable CSW with an optimization approach. Definition 5.11 shows the description of an optimization problem for finding the initial token values of a CSW. The example in Section 5.3 shows that the proposed criterion can be used to determine which initial token values or step sizes are better. The optimization approach is the basis for the automatic configuration algorithm (Algorithm 5.1) presented in Section 5.4.

Algorithm 5.1 is the final goal of this thesis. It is based on repeated co-simulation runs similar to the algorithm presented in [43]. The advantage of Algorithm 5.1 is that the underlying model of the system is more general. The co-simulation quality criterion does not have to have a physical meaning such as energy residual. Algorithm 5.1 tunes the co-simulation at multiple rates. This makes it possible to reduce the computational load on the platform that is used to run the co-simulation.

5.1 Connection defect

This section shows how to calculate the connection defect caused by a CSW (Theorem 5.2). The goal of defect control is to reduce the step size of the simulators so that the defect meets the tolerance. Theorem 5.4 shows that reducing the step sizes actually reduces the connection defect. This theorem also specifies which step sizes should be reduced in order to reduce the connection defect on a particular connection. It allows to take advantage of multi-rate co-simulation. With single-rate co-simulation, some of the simulators may need to be run with a larger step-size than required. A multi-rate calculation uses less processor time than a single-rate co-simulation.

Lemma 5.1 (Input signal)**.** *Let S be a CODESYS (Definition 2.2) and*

$$\begin{aligned} C &= \mathrm{systemToNetwork}\,(S, \mathrm{subsystemSolvers}) \\ G &= \mathrm{networkToDataflow}(C, \mathrm{h}, \mathrm{d}_0, \omega_0) \end{aligned} \tag{5.1}$$

The numerical solution for the input signal $\widetilde{u}_{\breve{\imath}\breve{\imath}} : \mathbb{R}_{>0} \to \mathcal{Z}$ *found by the CSW is*

$$\widetilde{u}_{\breve{\imath}\breve{\imath}}(t) = \begin{cases} \omega_0(\breve{b}, k_{\breve{\imath}}), & k_{\breve{\imath}} \leqslant r_{\breve{\imath}}, \quad k_{\breve{\imath}} = \left\lceil \frac{t}{\mathrm{h}(\breve{\imath})} \right\rceil \\ \gamma_{\hat{\imath}}(k_{\hat{\imath}}, \hat{o}, 1), & k_{\breve{\imath}} > r_{\breve{\imath}}, \quad k_{\hat{\imath}} = \left\lceil (k_{\breve{\imath}} - r_{\breve{\imath}}) \frac{\mathrm{h}(\breve{\imath})}{\mathrm{h}(\hat{\imath})} \right\rceil, \quad k_{\breve{\imath}} = \left\lceil \frac{t}{\mathrm{h}(\breve{\imath})} \right\rceil \end{cases} \tag{5.2}$$

where

- *$\hat{o} \in I_{\mathscr{Y}_{\hat{\imath}}}$ is the output port index of the simulator $A_{\hat{\imath}} = A(\hat{\imath})$,*
- *$\check{\imath} \in I_{\mathscr{U}_{\check{\imath}}}$ is the input port index of the simulator $A_{\check{\imath}} = A(\check{\imath})$,*
- *and the two ports are connected with buffers $\check{b}, \hat{b} \in I^{\mathrm{B}}$ and the rate converter $A_{\underline{a}} = \mathrm{A}(\underline{a})$ as described in* (3.141).

Proof. Lemma 3.12 shows how input tokens of an SDFA depend on the output tokens of the connected actor (3.81). From (3.81), (3.119), (3.132) and (4.23) it follows that input tokens of the rate converter $A_{\underline{\mathrm{a}}}$ are

$$\xi_{\underline{\mathrm{a}}}(k_{\underline{\mathrm{a}}}, 1, \check{\theta}_{\underline{\mathrm{a}}}) = \gamma_{\hat{\imath}}(k_{\hat{\imath}}, \hat{o}, 1), \quad k_{\hat{\imath}} = (k_{\underline{\mathrm{a}}} - 1)\,\mathrm{c}_{\underline{\mathrm{a}}}(1) + \check{\theta}_{\underline{\mathrm{a}}} \tag{5.3}$$

The token index in tokens produced by the simulator $A_{\hat{\imath}}$ can only be 1 due to (3.4) and (3.119). Lemma 3.13 enables the calculation to be written as (3.91). From (3.91) and (3.129) it follows that the output tokens of the rate converter $A_{\underline{\mathrm{a}}}$ are

$$\gamma_{\underline{\mathrm{a}}}(k_{\underline{\mathrm{a}}}, 1, \hat{\theta}_{\underline{\mathrm{a}}}) = \xi_{\underline{\mathrm{a}}}\left(k_{\underline{\mathrm{a}}}, 1, \left\lceil \frac{\hat{\theta}_{\underline{\mathrm{a}}}\,\mathrm{c}_{\underline{\mathrm{a}}}(1)}{\mathrm{p}_{\underline{\mathrm{a}}}(1)} \right\rceil\right) \tag{5.4}$$

where consumption and production rate of the rate converter are determined by (3.132). From (3.81), (3.118), (3.132) and (4.23) it follows that input tokens of the simulator $A_{\check{\imath}}$ are

$$\xi_{\check{\imath}}(k_{\check{\imath}}, \check{\imath}, 1) = \begin{cases} \omega_0(\check{b}, k_{\check{\imath}}), & k_{\check{\imath}} \leqslant r_{\check{\imath}} \\ \gamma_{\underline{\mathrm{a}}}(k_{\underline{\mathrm{a}}}, 1, \hat{\theta}_{\underline{\mathrm{a}}}), & \begin{array}{l} k_{\check{\imath}} > r_{\check{\imath}}, \quad k_{\check{\imath}} = r_{\check{\imath}} + (k_{\underline{\mathrm{a}}} - 1)\,\mathrm{p}_{\underline{\mathrm{a}}}(1) + \hat{\theta}_{\underline{\mathrm{a}}} \\ k_{\underline{\mathrm{a}}} = \left\lceil \frac{k_{\check{\imath}} - r_{\check{\imath}}}{\mathrm{p}_{\underline{\mathrm{a}}}(1)} \right\rceil \\ \hat{\theta}_{\underline{\mathrm{a}}} = k_{\check{\imath}} - r_{\check{\imath}} - (k_{\underline{\mathrm{a}}} - 1)\,\mathrm{p}_{\underline{\mathrm{a}}}(1) \end{array} \end{cases} \tag{5.5}$$

The token index in tokens consumed by the simulator $A_{\check{\imath}}$ can only be 1 due to (3.3) and (3.118). From (2.59), (3.120) and (3.122) it follows that

$$\widetilde{u}_{\check{\imath}\check{\imath}}(t) = \xi_{\check{\imath}}(k_{\check{\imath}}, \check{\imath}, 1), \quad k_{\check{\imath}} = \left\lceil \frac{t}{\mathrm{h}(\check{\imath})} \right\rceil \tag{5.6}$$

For $k_{\check{\imath}} \leqslant r_{\check{\imath}}$, the expression (5.2) follows from (5.5) and (5.6). From (5.3)-(5.6) it follows that for $k_{\check{\imath}} > r_{\check{\imath}}$

$$\widetilde{u}_{\check{\imath}\check{\imath}}(t) = \gamma_{\hat{\imath}}(k_{\hat{\imath}}, \hat{o}, 1) \tag{5.7}$$

where

$$k_{\check{\imath}} = \left\lceil \frac{t}{\mathrm{h}(\check{\imath})} \right\rceil, \quad \hat{\theta}_{\underline{\mathrm{a}}} = k_{\check{\imath}} - r_{\check{\imath}} - (k_{\underline{\mathrm{a}}} - 1)\,\mathrm{p}_{\underline{\mathrm{a}}}(1), \quad k_{\hat{\imath}} = (k_{\underline{\mathrm{a}}} - 1)\,\mathrm{c}_{\underline{\mathrm{a}}}(1) + \left\lceil \frac{\hat{\theta}_{\underline{\mathrm{a}}}\,\mathrm{c}_{\underline{\mathrm{a}}}(1)}{\mathrm{p}_{\underline{\mathrm{a}}}(1)} \right\rceil \tag{5.8}$$

From (5.8) it follows that

$$k_{\hat{\imath}} = \left\lceil (k_{\breve{\imath}} - r_{\breve{\imath}}) \frac{\mathrm{c}_{\underline{a}}(1)}{\mathrm{p}_{\underline{a}}(1)} \right\rceil, \quad k_{\breve{\imath}} = \left\lceil \frac{t}{\mathrm{h}(\breve{\imath})} \right\rceil \tag{5.9}$$

□

Theorem 5.2 (Connection defect of a CSW)**.** *Let S be a CODESYS (Definition 2.2) and*

$$\begin{aligned} C &= \text{systemToNetwork}\left(S, \text{subsystemSolvers}\right) \\ G &= \text{networkToDataflow}(C, \mathrm{h}, \mathrm{d}_0, \omega_0) \end{aligned} \tag{5.10}$$

Assume that

- *the initial tokens* d_0 *for the CSW are obtained using the method described in Definition 4.8*
- *and the CSW G is executed with a valid execution (Definition 3.7).*

The connection defect (2.37d) *of a numerical solution found by a CSW (Definition 3.25) is*

$$\delta\widetilde{u}_{\breve{\imath}\hat{\imath}}(t) = \begin{cases} \omega_0(\breve{b}, \breve{k}_{\breve{\imath}}) - \gamma_{\hat{\imath}}(\hat{k}_{\hat{\imath}}, \hat{o}, 1), & \breve{k}_{\breve{\imath}} \leqslant r_{\breve{\imath}}, \quad \breve{k}_{\breve{\imath}} = \left\lceil \frac{t}{\mathrm{h}(\breve{\imath})} \right\rceil, \quad \hat{k}_{\hat{\imath}} = \left\lceil \frac{t}{\mathrm{h}(\hat{\imath})} \right\rceil \\ \gamma_{\hat{\imath}}(\breve{k}_{\hat{\imath}}, \hat{o}, 1) - \gamma_{\hat{\imath}}(\hat{k}_{\hat{\imath}}, \hat{o}, 1), & \begin{array}{l} \breve{k}_{\breve{\imath}} > r_{\breve{\imath}}, \quad \breve{k}_{\breve{\imath}} = \left\lceil \frac{t}{\mathrm{h}(\breve{\imath})} \right\rceil, \\ \breve{k}_{\hat{\imath}} = \left\lceil (\breve{k}_{\breve{\imath}} - r_{\breve{\imath}}) \frac{\mathrm{h}(\breve{\imath})}{\mathrm{h}(\hat{\imath})} \right\rceil, \quad \hat{k}_{\hat{\imath}} = \left\lceil \frac{t}{\mathrm{h}(\hat{\imath})} \right\rceil \end{array} \end{cases} \tag{5.11}$$

Proof. From (2.60) and (3.123) it follows that

$$\widetilde{y}_{\hat{\imath}\hat{o}}(t) = \gamma_{\hat{\imath}}(\hat{k}_{\hat{\imath}}, \hat{o}, 1), \quad \hat{k}_{\hat{\imath}} = \left\lceil \frac{t}{\mathrm{h}(\hat{\imath})} \right\rceil \tag{5.12}$$

The expression (5.11) follows from Lemma 5.1, (2.37d) and (5.12). □

Theorem 5.2 shows how the connection defect introduced by a CSW can be calculated. This method can be used to evaluate the step sizes and initial token values, and enables automatic configuration, which is presented in the next chapter.

Theorem 5.4 shows how to control the connection defect. It shows that the connection defect can be reduced by reducing the communication step size of the connected simulators.

Lemma 5.3 (State signal in neighboring steps)**.** *Let M be a CODESUB (Definition 2.1) and*

$$\begin{aligned} F &= \text{subsystemToSlave}(M, \text{solver}) \\ A &= \text{slaveToSimulator}(F, h) \end{aligned} \tag{5.13}$$

Assume that

- *the state transition function* $\mathbf{f}$ *is Lipschitz continuous (Definition 2.3),*
- *the numerical solution for the input signal is bounded* $\|\widetilde{\mathbf{u}}(t)\| \leqslant B_{\widetilde{\mathbf{u}}}$*,*

- *the numerical solution for the state signal is bounded* $\|\widetilde{\mathbf{x}}(t)\| \leqslant B_{\widetilde{\mathbf{x}}}$,
- *and the integration defect is* $\delta\widetilde{\mathbf{x}}(t) = \mathscr{O}(h)$.

For $k, \Delta k \in \mathbb{N}$

$$\widetilde{\mathbf{x}}\big((k+\Delta k)h\big) - \widetilde{\mathbf{x}}(kh) = \mathscr{O}(h) \tag{5.14}$$

Proof. Since $\delta\widetilde{\mathbf{x}}(t) = \mathscr{O}(h)$ there exists $C, h_0 \in \mathbb{Q}_{>0}$ such that for all $h \leqslant h_0$

$$\|\delta\widetilde{\mathbf{x}}(t)\| \leqslant C\, h_0 \tag{5.15}$$

By the integration of (2.37a)

$$\widetilde{\mathbf{x}}\big((k+1)h\big) = \widetilde{\mathbf{x}}(kh) + \int\limits_{kh}^{(k+1)h} \mathbf{f}\big(\widetilde{\mathbf{x}}(\tau), \widetilde{\mathbf{u}}(\tau)\big) + \delta\widetilde{\mathbf{x}}(\tau)\, \mathrm{d}\tau \tag{5.16}$$

the following inequality is obtained for $h \leqslant h_0$

$$\big\|\widetilde{\mathbf{x}}\big((k+1)h\big)\big\| \leqslant \big\|\widetilde{\mathbf{x}}(kh)\big\| + \int\limits_{kh}^{(k+1)h} K_{\mathbf{f}}\, \big\|\widetilde{\mathbf{x}}(\tau)\big\| + K_{\mathbf{f}}\, B_{\widetilde{\mathbf{u}}} + C\, h\, \mathrm{d}\tau \tag{5.17}$$

The Gronwall lemma (Theorem 6 in [60]) shows that

$$\big\|\widetilde{\mathbf{x}}\big((k+1)h\big)\big\| - \big\|\widetilde{\mathbf{x}}(kh)\big\|\, e^{K_{\mathbf{f}} h} \leqslant \frac{K_{\mathbf{f}}\, B_{\widetilde{\mathbf{u}}} + C\, h}{K_{\mathbf{f}}} \left(e^{K_{\mathbf{f}} h} - 1\right) \tag{5.18}$$

From the Taylor expansion of the exponential function, $\|\widetilde{\mathbf{x}}(t)\| \leqslant B_{\widetilde{\mathbf{x}}}$ and

$$\big\|\widetilde{\mathbf{x}}\big((k+1)h\big)\big\| - \big\|\widetilde{\mathbf{x}}(kh)\big\| \leqslant \left(B_{\widetilde{\mathbf{x}}} + \frac{K_{\mathbf{f}}\, B_{\widetilde{\mathbf{u}}} + C\, h}{K_{\mathbf{f}}}\right) \sum_{n=1}^{\infty} (K_{\mathbf{f}}\, h)^n \tag{5.19}$$

it follows that

$$\big\|\widetilde{\mathbf{x}}\big((k+1)h\big)\big\| - \big\|\widetilde{\mathbf{x}}(kh)\big\| = \mathscr{O}(h) \tag{5.20}$$

The statement of the lemma follows by applying the above statement for Δk times recursively.

□

Theorem 5.4 (Connection defect bounds)**.** *Let*

- $S \in \mathcal{S}$ *be a CODESYS (Definition 2.2),*
- $C = \text{systemToNetwork}(S, \text{subsystemSolvers})$ *be its SYSW (Definition 2.16),*
- *and* $G = \text{networkToDataflow}(C, \mathrm{h}, \mathrm{d}_0, \omega_0)$ *be its CSW (Definition 3.25).*

Assume that

- *the numerical solution for the input signals is bounded* $\|\widetilde{\mathbf{u}}_i(t)\| \leqslant B_{\widetilde{\mathbf{u}}}$,
- *the numerical solution for the state signal is bounded* $\|\widetilde{\mathbf{x}}_i(t)\| \leqslant B_{\widetilde{\mathbf{x}}}$,

- *the integration defects are* $\delta\widetilde{\mathbf{x}}_i(t) = \mathscr{O}\big(\mathrm{h}(i)\big)$*,*
- *the state transition function* $\mathbf{f}_i$ *of each CODESUB is Lipschitz continuous (Definition 2.3), the output function* $\mathbf{g}_i$ *of each CODESUB is Lipschitz continuous and does not depend of the CODESUB inputs*

$$\mathbf{g}_i\big(\widetilde{\mathbf{x}}_i(t), \widetilde{\mathbf{u}}_i(t)\big) = \mathbf{g}_i\big(\widetilde{\mathbf{x}}_i(t)\big), \quad i \in I_\mathrm{M} \tag{5.21}$$

and the step sizes of connected simulators $\hat\imath, \check\imath \in I_\mathrm{F}$ *are*

$$\mathrm{h}(\hat\imath) = q_{\hat\imath}\, h_{\hat\imath\check\imath}, \quad \mathrm{h}(\check\imath) = q_{\check\imath}\, h_{\hat\imath\check\imath}, \quad q_{\hat\imath}, q_{\check\imath}, h_{\hat\imath\check\imath} \in \mathbb{Q}_{>0} \tag{5.22}$$

For $t > H$*, the connection defect* (2.37d) *of a numerical solution found by a CSW (Definition 3.25) is*

$$\delta\widetilde{u}_{\check\imath\hat\imath}(t) = \mathscr{O}(h_{\hat\imath\check\imath}) \tag{5.23}$$

Proof. From Theorem 5.2, (2.60), (2.73), (3.123), (5.21) and (5.22)

$$\delta\widetilde{u}_{\check\imath\hat\imath}(t) = g_{\hat\imath\hat o}\big(\widetilde{\mathbf{x}}_{\hat\imath}(\hat t)\big) - g_{\hat\imath\hat o}\big(\widetilde{\mathbf{x}}_{\hat\imath}(\check t)\big) \tag{5.24}$$

where

$$\hat t = \hat k_{\hat\imath}\, \mathrm{h}(\hat\imath), \quad \check t = \check k_{\hat\imath}\, \mathrm{h}(\hat\imath) \tag{5.25}$$

and

$$\check k_{\check\imath} = \left\lceil \frac{t}{\mathrm{h}(\check\imath)} \right\rceil, \quad \check k_{\hat\imath} = \left\lceil (\check k_{\check\imath} - r_{\check\imath}) \frac{h(\check\imath)}{h(\hat\imath)} \right\rceil, \quad \hat k_{\hat\imath} = \left\lceil \frac{t}{\mathrm{h}(\hat\imath)} \right\rceil \tag{5.26}$$

(5.26) can be reformulated to the system of inequalities

$$\check k_{\check\imath} - 1 < \frac{t}{\mathrm{h}(\check\imath)} \leqslant \check k_{\check\imath}, \quad \check k_{\hat\imath} - 1 < (\check k_{\check\imath} - r_{\check\imath}) \frac{h(\check\imath)}{h(\hat\imath)} \leqslant \check k_{\hat\imath}, \quad \hat k_{\hat\imath} - 1 < \frac{t}{\mathrm{h}(\hat\imath)} \leqslant \hat k_{\hat\imath} \tag{5.27}$$

From (5.22) and (5.27) it follows that

$$\hat k_{\hat\imath} - \check k_{\hat\imath} < r_{\check\imath} \frac{q_{\check\imath}}{q_{\hat\imath}} + 1 \tag{5.28}$$

Since $\mathbf{g}_{\hat\imath}$ is Lipschitz continuous, from (5.24) it follows that

$$\left\| g_{\hat\imath\hat o}\big(\widetilde{\mathbf{x}}_{\hat\imath}(\hat t)\big) - g_{\hat\imath\hat o}\big(\widetilde{\mathbf{x}}_{\hat\imath}(\check t)\big) \right\| \leqslant K_{\mathbf{g}_{\hat\imath}} \left\| \widetilde{\mathbf{x}}_{\hat\imath}(\hat t) - \widetilde{\mathbf{x}}_{\hat\imath}(\check t) \right\| \tag{5.29}$$

The statement of the theorem follows from Lemma 5.3, (5.24), (5.28) and (5.29). □

5.2 Output defect estimation

Theorem 2.17 shows that the output defect can be controlled by the step size of the simulator. However, the output defect cannot be obtained directly without knowing the internal equations and signals from the CODESUB. This section introduces defect estimation wrapper (Definition 5.6) for the purpose of output defect estimation. Theorem 5.9 shows that such an output defect estimation is asymptotically correct.

Definition 5.5 (Output defect estimator)**.** An output defect estimator (ODEST) is a tuple

$$(A', \zeta) = \big(I_{\mathscr{U}}, \mathscr{U}, I_{\mathscr{Y}'}, \mathscr{Y}', \mathrm{c}, \mathrm{p}, \mathrm{calculate}'\big) = \mathrm{slaveToEstimator}\,(F, h) \tag{5.30}$$

where

- F is a CSL (Definition 2.13) that solves a CODESUBs (Definition 2.1),
- $h \in \mathbb{Q}_{>0}$ is the step size of the simulator (Definition 3.23),
- slaveToEstimator : $\mathcal{F} \times \mathbb{Q}_{>0} \to \mathcal{A} \times \mathcal{Z}^{\mathbb{N} \times I_{\mathrm{Y}}}$ is the function that constructs the simulator,
- $\zeta : \mathbb{N} \times I_{\mathrm{Y}} \to \mathcal{Z}$ is the sequence of intermediate values produced by the ODEST,

$$A = (I_{\mathscr{U}}, \mathscr{U}, I_{\mathscr{Y}}, \mathscr{Y}, \mathrm{c}, \mathrm{p}, \mathrm{calculate}) = \mathrm{slaveToSimulator}\,(F, h) \tag{5.31}$$

is the simulator (Definition 3.23) used to execute F and

$$A' = (I_{\mathscr{U}}, \mathscr{U}, I_{\mathscr{Y}}, \mathscr{Y}, \mathrm{c}, \mathrm{p}, \mathrm{calculate}') \tag{5.32}$$

is an SDFA (Definition 3.1). The function calculate$'$ (3.2) performs two state updates. First, the input tokens (2.57) are assigned as input values of the CSL

$$u'(2k-1, \imath) = \xi'(k, \imath), \quad \imath \leqslant N_{\mathrm{U}} \tag{5.33}$$

and the state of the CSL

$$v'(2k-2) = \xi'(k, N_{\mathscr{U}}) \tag{5.34}$$

and then the state of the CSL is updated according to (2.74) with the step size halved

$$v'(2k-1) = \mathrm{doStep}\left(v'^{N_{\mathrm{U}}}(2k-1), \frac{h}{2}\right) \tag{5.35}$$

The sequence of intermediate values is updated

$$\zeta(k, o) = \mathrm{get}\big(o, v'(2k-1)\big), \quad o < N_{\mathscr{Y}} \tag{5.36}$$

Before the state update the input tokens are assigned as input values of the CSL

$$u'(2k,\iota) = \xi'(k,\iota), \quad \iota \leqslant N_{\mathrm{U}} \tag{5.37}$$

The state of the CSL is updated according to (2.74) with the step size halved

$$v'(2k) = \mathrm{doStep}\left(v'^{N_{\mathrm{U}}}(2k), \frac{h}{2}\right) \tag{5.38}$$

The output tokens for the rest of the ports are generated

$$\gamma'(k,o,1) = \begin{cases} \mathrm{get}\big(o, v'(2k)\big), & o \in I_{\mathrm{Y}} \\ v'(2k), & o = N_{\mathscr{Y}} \end{cases} \tag{5.39}$$

Definition 5.6 (Defect estimation wrapper)**.** Let C be a CSN (Definition 2.61) and

$$G = \left(I^{\mathrm{A}}, \mathrm{A}, I^{\mathrm{B}}, \mathrm{src}, \mathrm{dst}, \mathrm{d}_0, \omega_0\right) = \mathrm{networkToDataflow}(C, \mathrm{h}, \mathrm{d}_0, \omega_0) \tag{5.40}$$

A defect estimation wrapper (DEW) is an SDFG

$$G' = \left(I^{\mathrm{A}}, \mathrm{A}', I^{\mathrm{B}}, \mathrm{src}, \mathrm{dst}, \mathrm{d}_0, \omega_0\right) = \mathrm{networkToEstimation}(C, \mathrm{h}, \mathrm{d}_0, \omega_0) \tag{5.41}$$

where

$$\mathrm{A}'(a) = \begin{cases} \mathrm{slaveToEstimator}\big(\mathrm{F}(i), \mathrm{h}(i)\big), & a = i, \quad i \in I_{\mathrm{F}} \\ \mathrm{A}(a), & a > N_{\mathrm{F}} \end{cases} \tag{5.42}$$

Definition 5.7 (Output signal interpolation)**.** Let

- $M \in \mathcal{M}$ be a CODESUB (Definition 2.1),
- $F = \mathrm{subsystemToSlave}(M, \mathrm{solver})$ be its SUBW (Definition 2.15),
- and $A = \mathrm{slaveToEstimator}(F, h)$ be its ODEST (Definition 5.5).

The numerical solution for the output signal of the CODESUB found by an ODEST (Definition 5.5) is equal to

$$\widetilde{y}'_o(t) = \gamma'(k,o,1), \quad (k-1)h < t \leqslant kh, \quad o \leqslant N_{\mathbf{Y}} \tag{5.43}$$

The output signal generated at all output ports of the CODESUB is equal to

$$\widetilde{\mathbf{y}}'(t) = \begin{bmatrix} \widetilde{y}'_1(t) & \widetilde{y}'_2(t) & \dots & \widetilde{y}_{N_{\mathrm{A}'}}(t) \end{bmatrix}^T \tag{5.44}$$

For the purpose of estimating the output defect between the communication points, a Hermite interpolation polynomial [76] is introduced next.

Definition 5.8 (Hermite interpolation polynomial)**.** Let

- $M \in \mathcal{M}$ be a CODESUB (Definition 2.1),
- $F = \text{subsystemToSlave}(M, \text{solver})$ be its SUBW (Definition 2.15),
- and $A = \text{slaveToEstimator}(F, h)$ be its ODEST (Definition 5.5).

The Hermite interpolation polynomial of an output signal generated by an ODEST (Definition 5.5) is equal to

$$\hat{y}_o(t) = \widetilde{y}'_o(t) + \frac{kh - t}{0.5h}\left[\zeta(k,o) - \widetilde{y}'_o(t)\right], \quad (k-1)h < t \leqslant kh \tag{5.45}$$

The Hermite interpolation interpolation polynomial of a signal generated at all output ports of the CODESUB is equal to

$$\hat{\mathbf{y}}(t) = \begin{bmatrix} \hat{y}_1(t) & \hat{y}_2(t) & \dots & \hat{y}_{N_{A'}}(t) \end{bmatrix}^T \tag{5.46}$$

A Hermite interpolation polynomial is consistent with multiple samples of the signals and their derivatives. The polynomial used in this thesis is consistent with signal values at two communication points and signal derivatives at the later point

$$\begin{aligned} &\hat{\mathbf{y}}(t(k) - \frac{h(k)}{2}) = \mathbf{g}_i\left(\widetilde{\mathbf{x}}_i(t(k) - \frac{h(k)}{2}), \widetilde{\mathbf{u}}_i(t(k) - \frac{h(k)}{2})\right) \\ &\hat{\mathbf{y}}(t(k)) = \mathbf{g}_i\left(\widetilde{\mathbf{x}}_i(t(k)), \widetilde{\mathbf{u}}_i(t(k))\right) \end{aligned} \tag{5.47}$$

The Hermite interpolation polynomial is used to obtain an asymptotically correct estimate of the output defect.

Theorem 5.9 (Estimate of the output defect)**.** *The estimation of the output defect is defined as the difference between interpolation polynomials* (5.45) *and* (5.43)

$$\hat{\delta}\widetilde{\mathbf{y}}'(t) = \widetilde{\mathbf{y}}'(t) - \hat{\mathbf{y}}(t) \tag{5.48}$$

Suppose the function $\mathbf{g}$ *is continuously differentiable and*

$$\widetilde{\mathbf{x}}'(t) = \widetilde{\mathbf{x}}'(kh) + \frac{\mathrm{d}\widetilde{\mathbf{x}}'}{\mathrm{dt}}(kh)(t - kh) + \mathscr{O}\left(h^2\right) \tag{5.49}$$

Then the estimate of the output defect (5.48) *is asymptotically correct, i.e. for each* $\alpha \in (0, 1]$

$$\lim_{h \to 0} \frac{\hat{\delta}\widetilde{\mathbf{y}}'(t)}{\delta\widetilde{\mathbf{y}}'(t)} = 1, \quad t = (k-1)h + \alpha h \tag{5.50}$$

Proof. Since $\mathbf{g}$ is continuously differentiable, it is also Lipschitz continuous. Lipschitz continuity (Definition 2.3) and (5.49) imply

$$\mathbf{g}\left(\widetilde{\mathbf{x}}'(t),\widetilde{\mathbf{u}}'(t)\right) = \mathbf{g}\left(\widetilde{\mathbf{x}}'(t),\widetilde{\mathbf{u}}'(t)\right)(kh) + \left.\frac{\mathrm{d}\mathbf{g}\left(\widetilde{\mathbf{x}}'(\tau),\widetilde{\mathbf{u}}'(\tau)\right)}{\mathrm{d}\tau}\right|_{\tau=kh}(t-kh) + \mathscr{O}\left(h^2\right) \tag{5.51}$$

The output defect (2.4b) on the interval $((k-1)\,h, kh]$ is equal to

$$\delta\widetilde{\mathbf{y}}'(t) = \widetilde{\mathbf{y}}'(t) - \mathbf{g}\left(\widetilde{\mathbf{x}}'(t),\widetilde{\mathbf{u}}'(t)\right) = \left.\frac{\mathrm{d}\mathbf{g}\left(\widetilde{\mathbf{x}}'(t),\widetilde{\mathbf{u}}'(t)\right)}{\mathrm{d}t}\right|_{t=kh}(t-kh) + \mathscr{O}\left(h^2\right) \tag{5.52}$$

The estimate of the output defect on the interval $(k-1)\,h < t \leqslant kh$ is equal to

$$\begin{aligned}\hat{\delta}\widetilde{\mathbf{y}}'(t) &= \frac{kh-t}{0.5h}\left[\mathbf{g}\left(\widetilde{\mathbf{x}}'(kh-\frac{h}{2}),\widetilde{\mathbf{u}}'(kh-\frac{h}{2})\right) - \widetilde{\mathbf{y}}'(kh-\frac{h}{2})\right] \\ &= \frac{kh-t}{0.5h}\left[\left.\frac{\mathrm{d}\mathbf{g}\left(\widetilde{\mathbf{x}}'(t),\widetilde{\mathbf{u}}'(t)\right)}{\mathrm{d}t}\right|_{t=kh}\frac{-h}{2} + \mathscr{O}\left(h^2\right)\right] \\ &= \left.\frac{\mathrm{d}\mathbf{g}\left(\widetilde{\mathbf{x}}'(t),\widetilde{\mathbf{u}}'(t)\right)}{\mathrm{d}t}\right|_{t=kh}(t-kh) + \mathscr{O}\left(h^2\right)\end{aligned} \tag{5.53}$$

The equation (5.50) follows directly from (5.52) and (5.53). □

5.3 Comparison of co-simulation wrappers

Model-based development examples in Section 4.3 use CSWs to facilitate testing of a control algorithm. In such a case, it is sufficient to use the same SDFG in the respective MIL and HIL simulations for testing and identification purposes. This enables the analysis of the differences between the modified simulators. However, if the coupling between the simulators is mechanical, the co-simulation can have large errors in predicting actual system behavior. This is one of the main reasons to choose such a system as the test system for co-simulation in many research papers [15, 31, 39, 53, 54, 55]. This section shows how to use the quality criterion presented in Definition 5.10 to compare DEWs

The proposed criterion enables an optimization approach to find initial token values of a DEW (Definition 5.11). This procedure provides a default option for determining the initial token values and is part of the automatic configuration algorithm presented in the next section. To check the validity of the proposed approach, three different CSWs are compared in Example 5.12. The values of the criterion are compared with the numerical error of individual signals.

Definition 5.10 (Quality criterion)**.** The criterion for the quality evaluation of DEW parameters

(Definition 5.6) is

$$J\big(C,\mathrm{h},\mathrm{d}_0,\omega_0,t_{end}\big) = \max_{0<t\leqslant t_{end}} \big(\|\delta\widetilde{\mathbf{U}}'(t)\|_\infty, \|\hat{\delta}\widetilde{\mathbf{Y}}'(t)\|_\infty\big) \tag{5.54}$$

where

- C is a CSN (Definition 2.61),
- h : $I_\mathrm{F} \to \mathbb{Q}_{>0}$ is the function assigning step-sizes of the simulators,
- d $_0 : I^\mathrm{B} \to \mathbb{N}_0$ are numbers of initial tokens (Definition 3.2)
- $\omega_0 : I^\mathrm{B} \to \mathbb{R}^\mathbb{N}$ are values of initial tokens (Definition 3.2),
- $t_{end} \in \mathbb{Q}_{>0}$ is the duration of the co-simulation experiment,
- $\delta\widetilde{\mathbf{U}}'$ is the connection defect obtained by aggregation (2.7) of the connection defect (Theorem 5.2) between ODESTs (Definition 5.5)
- $\hat{\delta}\widetilde{\mathbf{Y}}'$ is the estimate of the output error obtained by aggregation (2.7) of the output defect estimates (Theorem 5.9) using ODESTs.

The method described in Definition 4.8 can be used by default to determine the number of initial tokens in a CSW. The optimization approach described below can be used as a default approach to determine the initial token values.

Definition 5.11 (Values of initial tokens)**.** Assume that the CSN C (Definition 2.14) should be executed in a co-simulation experiment. Assume that simulator step sizes $\mathrm{h} : I_\mathrm{F} \to \mathbb{Q}_{>0}$ and the number of initial tokens $\mathrm{d}_0 : I^\mathrm{B} \to \mathbb{N}_0$ are given. The values of initial tokens can be determined by the minimization of the quality criterion (Definition 5.10)

$$\begin{aligned}\omega^H_{0opt} &= \mathrm{findTokenValues}(C,\mathrm{h},\mathrm{d}_0)\\ &= \arg \min_{\omega_0' : I^\mathrm{B} \to \mathbb{R}^\mathbb{N}} J\big(C,\mathrm{h},\mathrm{d}_0,\omega_0',H\big)\end{aligned} \tag{5.55}$$

in the experiment lasting a single iteration (Theorem 3.29).

The above definition implies that the problem of finding initial tokens can be solved through optimization. The concrete implementation used in examples can be found in the Python procedure `sdf4sim.autoconfig.find_initial_tokens` in the repository [44]. An implementation of the Nelder-Mead algorithm [77, 78] was used to solve the optimization problem. This is an iterative algorithm which starts the optimization at the provided guess. The tokens provided to the algorithm were obtained by running a single iteration of the SDFG with null tokens. The resulting output tokens were suggested as the initial guess for the algorithm.

The next example shows the co-simulation of a two-mass oscillator in which two oscillators are coupled with a mechanical damper. Three DEWs are compared based on the proposed criterion. This comparison shows the applicability of the criterion for an optimization-based approach to the configuration of DEWs.

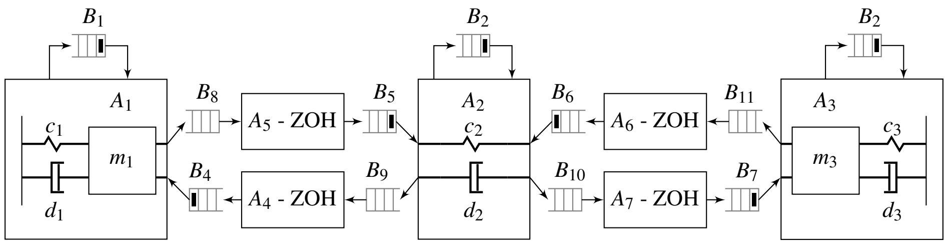


Figure 5.2: The CSWs presented in Example 5.12 have the same number of initial tokens. The differences between them are in initial token values and simulator step sizes.

Example 5.12 (Comparison of three CSWs)**.** Example 2.7 shows the CODESYS modeling a two-mass oscillator using three subsystems. This CODESYS is wrapped with the SYSW C shown in Example 2.19. This example introduces three CSWs for the given SYSW. The number of initial tokens for the three CSWs is selected using the method introduced in Definition 4.8. In each CSW all simulator step sizes are equal which results in the same number of initial tokens (Figure 5.2). The first CSW

$$G_1 = \text{networkToDataflow}(C, \text{h}_1, \text{d}_{01}, \omega_{01}) \tag{5.56}$$

has the simulator step sizes set to

$$\text{h}_1(i) = \frac{1}{4}, \quad i \in I_\text{F} \tag{5.57}$$

uses the method presented in Definition 4.8 to find the number of initial tokens $\text{d}_{01} : I^\text{B} \to \mathbb{N}_0$ and the method presented in Definition 5.11 to find the values of initial tokens $\omega_{01} : I^\text{B} \to \mathcal{Z}^\mathbb{N}$

$$\text{d}_0, \omega_0 \tag{5.58}$$

The second CSW

$$G_2 = \text{networkToDataflow}(C, \text{h}_2, \text{d}_{02}, \omega_{02}) \tag{5.59}$$

has the simulator step sizes set to

$$\text{h}_2(i) = \frac{1}{2}, \quad i \in I_\text{F} \tag{5.60}$$

uses the method presented in Definition 4.8 to find the number of initial tokens $\text{d}_{02} : I^\text{B} \to \mathbb{N}_0$ and the method presented in Definition 5.11 to find the values of initial tokens

$$\omega_{02} = \text{findTokenValues}\big(C, \{\text{h}_2\} \times \{\text{d}_{02}\} \times (I^\text{B} \to \mathcal{Z}^\mathbb{N}), tol, H\big) \tag{5.61}$$

The third CSW

$$G_3 = \text{networkToDataflow}(C, \text{h}_3, \text{d}_{03}, \omega_{03}) \tag{5.62}$$

CSW	Quality criterion (5.54)	Force error (5.65)	Speed error (5.66)
(5.57)	0.073	0.041	0.006
(5.59)	0.123	0.088	0.013
(5.62)	1.000	0.476	0.014

Table 5.1: Comparison of analogous terms

has the simulator step sizes set to

$$\mathrm{h}_3(i) = \frac{1}{4}, \quad i \in I_\mathrm{F} \tag{5.63}$$

uses the method presented in Definition 4.8 to find the number of initial tokens $\mathrm{d}_{02} : I^\mathrm{B} \to \mathbb{N}_0$ and values of initial tokens set to

$$\omega_{03}(b,1) = 0, \quad \mathrm{dst}(b) = (a,\imath), \quad a = i, \quad i \in I_\mathrm{F}, \quad \imath \in I_{\mathrm{U}_i} \tag{5.64}$$

Their responses are compared with a monolithic solution of the CODESYS obtained using the Python solver `scipy.integrate.solve_ivp`. The Python code to execute the example is available [44] in the procedure `sdf4sim.example.twomass.three_cosimulations_comparison`. The criterion introduced in Definition 5.10 is evaluated for each CSW. Table 5.1 shows the val-

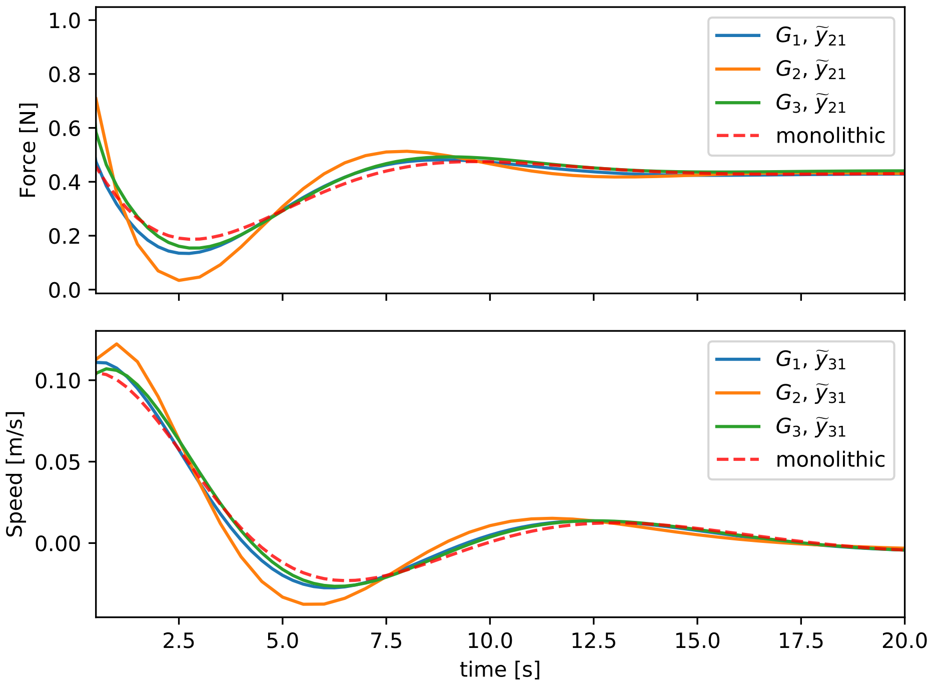


Figure 5.3: Signal responses obtained from CSWs shown in Example 5.12.

ues of this quality criterion, the force error

$$\max_{0<k\mathrm{h}(2)\leqslant t_{end}} |\Delta\widetilde{y}_{21}| \approx \max_{0<k\mathrm{h}(2)\leqslant t_{end}} \left|\gamma_2(k,1,1) - \bar{y}_{21}\big(k\mathrm{h}(2)\big)\right| \tag{5.65}$$

and the speed error

$$\max_{0<k\mathrm{h}(3)\leqslant t_{end}} |\Delta\widetilde{y}_{31}| \approx \max_{0<k\mathrm{h}(3)\leqslant t_{end}} \left|\gamma_3(k,1,1) - \bar{y}_{21}\big(k\mathrm{h}(3)\big)\right| \tag{5.66}$$

5.4 Automatic configuration

In this section Algorithm 5.1 for the automatic configuration of the co-simulation is presented. The presented algorithm brings together the information from the entire thesis (Figure 1.3). The algorithm finds the step sizes and initial tokens for the DEW (Definition 5.6) in such a way that the defect tolerance is met during the co-simulation experiment. Theorem 5.2 shows how initial tokens affect the connection defect. Theorem 5.4 shows that the connection defect can be reduced by reducing the communication step size of connected simulators. Theorem 2.17 shows that the output defect can be reduced by reducing the communication step size of the simulator. The method for determining the number of initial tokens is described in Definition 4.8. The previous section showed how an optimization approach can be used to find initial token values (Definition 5.11). After searching for initial tokens, the step sizes are adjusted (Algorithm 5.14). Algorithm 5.1 iteratively repeats the search for initial tokens and the adjustment of step sizes until the tolerance is reached.

Definition 5.13 (ODEST defect)**.** Let

- $S \in \mathcal{S}$ be a CODESUB (Definition 2.2),
- $C = \mathrm{systemToNetwork}(S, \mathrm{subsystemSolvers})$ be its SYSW (Definition 2.16),
- $G = \mathrm{networkToEstimation}(C, \mathrm{h}, \mathrm{d}_0, \omega_0)$ be its DEW (Definition 5.6),
- and $t_{end} \in \mathbb{Q}_{>0}$ be the duration of the co-simulation experiment.

Assume that the co-simulation experiment with the DEW G and the duration $t_{end} \in \mathbb{Q}_{>0}$ is performed. The ODEST defect is equal to

$$\delta\mathrm{A}(i, t_{end}) = \max_{0<t\leqslant t_{end}} \max \begin{pmatrix} \max\limits_{\iota\leqslant N_{\mathbf{u}_i}} \|\delta\widetilde{u}_{i\iota}(t)\|, \\ \max\limits_{\substack{o\leqslant N_{\mathbf{y}_i},\\ (\check{\imath},\check{\iota})=\mathrm{L}(i,o)}} \|\delta\widetilde{u}_{\check{\imath}\check{\iota}}(t)\|, \\ \max\limits_{o\leqslant N_{\mathbf{y}_i}} \|\delta\widetilde{y}_{io}(t)\|, \end{pmatrix} \tag{5.67}$$

The previous definition introduces the ODEST defect to reformulate the connection (2.37d) and the output defect (2.37b). The quality criterion (Definition 5.10) can be reformulated to

$$J\big(C,\mathrm{h},\mathrm{d}_0,\omega_0,t_{end}\big) = \max_{0<t\leqslant t_{end}} \max_{i\in I_\mathrm{F}} \delta\mathrm{A}(i,t_{end}) \tag{5.68}$$

Such a reformulation makes it possible to individually adapt the step sizes of the simulator. Theorem 2.17 and Theorem 5.4 show that the simulator step size influences the respective ODEST defect. This justifies using the following method to adjust simulator step sizes.

Definition 5.14 (Step size adjustment)**.** The function that adjusts the step sizes of a DEW to meet the defect tolerance is

$$\mathrm{adaptStepSizes} : \mathcal{G} \times \mathbb{Q}_{>0}{}^{I_\mathrm{F}} \times \mathbb{N}_0{}^{I_\mathrm{F}} \times \big(I^\mathrm{B} \to \mathcal{Z}^\mathbb{N}\big) \times \mathbb{Q}_{>0} \times \mathbb{Q}_{>0} \to \mathbb{Q}_{>0}{}^{I_\mathrm{F}} \tag{5.69}$$

The adjusted step sizes are obtained by

$$\mathrm{h}' = \mathrm{adaptStepSizes}(G,t_{end},tol) \tag{5.70}$$

where

- G is the DEW (Definition 5.6,)
- $\mathrm{h}' : I_\mathrm{F} \to \mathbb{Q}_{>0}$ are the step sizes after the adjustment,
- $\mathrm{h} : I_\mathrm{F} \to \mathbb{Q}_{>0}$ are the step sizes before the adjustment,
- $t_{end} \in \mathbb{Q}_{>0}$ is the duration of the co-simulation experiment,
- and $tol \in \mathbb{Q}_{>0}$ is the defect tolerance.

The adjusted step sizes are

$$\mathrm{h}'(i) = \begin{cases} \frac{\mathrm{h}(i)}{2^n}, & \delta\mathrm{A}(i,t_{end}) > tol, \quad n = \left\lceil \log_2\left(\frac{\delta\mathrm{A}(i,t_{end})}{tol}\right)\right\rceil \\ \mathrm{h}(i), & \delta\mathrm{A}(i,t_{end}) \leqslant tol \end{cases} \tag{5.71}$$

with the ODEST defect (Definition 5.13) which estimated using the DEW (Definition 5.6)

$$G = \mathrm{networkToEstimation}(C,\mathrm{h},\mathrm{d}_0,\omega_0) \tag{5.72}$$

An implementation of the method defined above is implemented using the Python procedure `sdf4sim.autoconfig._step_reduction_factor` [44]. The methods introduced in Definition 4.8, Definition 5.11 and Definition 5.14 form the building blocks of Algorithm 5.1. The method for finding initial tokens (Definition 4.8) enables a check to be made as to whether the resulting CSW can be executed in real time (Theorem 4.17). The method for calculating initial values (Definition 5.11) should ensure the optimal initial token values for the first SDFI. The step size adjustment (Definition 5.14) should ensure that the algorithm finds the parameters in

Algorithm 5.1 The search for suitable DEW parameters

```
Require: $C \in \mathcal{C}, \quad t_{end} \in \mathbb{Q}_{>0}, \quad tol \in \mathbb{R}_{>0}, \quad h_{init} \in \mathbb{Q}_{>0}$
  for $i \in I_{\mathrm{F}}$ do
      $\mathrm{h}(i) := h_{init}$
  do
      $\mathrm{d}_0 := \mathrm{findTokenNumbers}(C, \mathrm{h})$
      $\omega_0 := \mathrm{findTokenValues}(C, \mathrm{h}, \mathrm{d}_0)$
      $G := \mathrm{networkToEstimation}(C, \mathrm{h}, \mathrm{d}_0, \omega_0)$
      $\mathrm{h} := \mathrm{adaptStepSizes}(G, t_{end}, tol)$
  while $J\big(C, \mathrm{h}, \mathrm{d}_0, \omega_0, t_{end}\big) > tol$
```

a small number of repeated executions. Algorithm 5.1 is the end result of this thesis. It gives a user a procedure to get CSW parameters to run a CSN with the specified quality.

Example 5.15 (Two-mass oscillator)**.** Example 2.7 shows the CODESYS modeling a two-mass oscillator using three subsystems. This CODESYS is wrapped with the SYSW C shown in Example 2.19. This example uses Algorithm 5.1 find three DEWs given different tolerances. The tolerances are listed in Table 5.2. The same table shows the actual values of the quality criterion (5.54) and the numerical error in individual signals. The comparison made is similar to the one shown in Example 5.12. The signal responses of the analyzed signals are shown in Figure 5.4. The implementation of this example can be found in the Python procedure `sdf4sim.twomass.example.three_tolerances_auto`, which is available at [44].

CSW	tol	Quality criterion (5.54)	Force error (5.65)	Speed error (5.66)
(5.57)	0.9	0.199	0.150	0.008
(5.59)	0.3	0.058	0.102	0.002
(5.62)	0.1	0.015	0.083	0.003

Table 5.2: Comparison of DEWs obtained with Algorithm 5.1

The previous example automates the trial and error procedure introduced in Example 5.12. A DEW is found for three different values of the requested tolerance. The three DEWs are compared in Table 5.2 and Figure 5.4. The example is important because coupling between the systems is mechanical. In such a case, a discrete model can only approximate the behavior. The quality of the approximation is influenced by the requested tolerance. Theorems 2.12, 2.17 and 5.4 guarantee that the co-simulation error is small when the CSLs are adequately solved and the tolerance is low. Since the solver in the example is analytical (2.90), only the output and connection defects introduced contribute to the co-simulation error. The ability of Algorithm 5.1 to reduce the co-simulation error in such a case is confirmed by the results observed in Table5.2

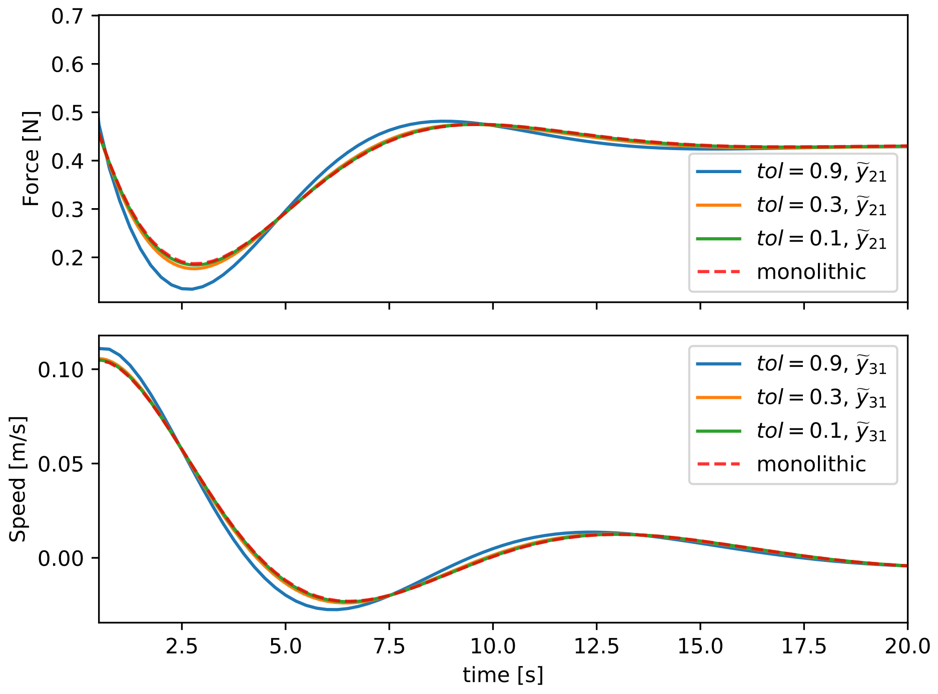


Figure 5.4: Responses from SYSWs obtained through the automatic configuration, taking into account the requested tolerances in Example 5.15.

and Figure 5.4. A bad solver can always lead to an increase in the co-simulation error. However, one of the main advantages of using co-simulations in practice is that a solver can be carefully tailored to the CSL. With suitable slaves, Algorithm 5.1 can configure the non-iterative co-simulation in order to achieve the desired quality.

Chapter 6

Conclusion

6.1 Contributions of the thesis

Contribution 1. A synchronous data flow model for execution of a non-iterative co-simulation.

CSW is introduced in Definition 3.25 as the model of computation for non-iterative co-simulation. Each CSWs is an SDFGs (Definition 3.2) with consistent consumption and production rates (Theorem 3.28). Theorem 3.29 shows that the proposed formalism correctly updates the simulated time in each CSL (Definition 2.13) of the simulated CSN (Definition 2.14). Examples 3.30 and 5.12 show CSWs that model the execution of a simple control loop and a two-mass oscillator.

An SDFG is a deterministic model of computation [34]. Section 3.2 is introduced to revisit the proof of determinacy due to its importance. Determinacy has the practical advantage that co-simulation results can be reproduced on different platforms and with different algorithms (Algorithms 3.1 and 4.2). Further practical benefit can be observed in Example 4.20. This example shows how such an algorithm can be used for test-driven development.

An additional advantage of modeling a non-iterative co-simulation with an SDFG is the available scheduling research [35, 36, 65]. Existing research results enable platform developers to optimize their implementation for the platform under their responsibility. The results should be the same for any valid implementation, although the execution time or memory consumption does not have to be.

To the best of the author's knowledge, Contribution 1 is an original contribution of this thesis. In [37] there is an example of wrapping an SDFG with an FMU. That work inspired this thesis to wrap a network of FMUs with an SDFG. CSW (Definition 3.25) is a generalized version of existing non-iterative Gauss-Seidel, Jacobi and multi-rate masters [2, 31, 33].

Contribution 2. A method for practical evaluation of co-simulation quality using connection and output defect.

The connection defect of two connected simulators in a CSW can be directly calculated (Theorem 5.2). ODEST is introduced in Definition 5.5 as a modified simulator for a CSL (Definition 2.13). An ODEST provides the ability to estimate the output defect. The method for estimating the output defect is asymptotically correct (Theorem 5.9). The connection defect and the output defect are aggregated into co-simulation quality criterion introduced in Definition 5.10. Examples 5.12 and 5.15 show the comparison between the values of the criterion and the error of individual signals in the co-simulation.

The co-simulation quality criterion is used to define an optimization approach for finding initial token values of a CSW (Definition 5.11). Such a method relieves a co-simulation user from having to set the initial tokens. In Algorithm 5.1 this criterion is also used to determine the stopping criterion. This approach is justified by Theorem 2.12 which shows that co-simulation error is bounded by connection, output and integration defects. The integration defect is left in the responsibility of the internal CSL solvers due to black box nature of co-simulation (Figure 5.1).

To the best of the author's knowledge, Contribution 2 is an original contribution of this thesis. The existing co-simulation error analysis is based on the local error analysis [39, 40]. The work presented in this thesis is based on numerical defect analysis [47, 61]. This thesis is influenced by the work on differential-algebraic equations [41]. The differences are the result of the black box nature of the co-simulation. For this reason, the algebraic equations of the underlying CODESYS (Definition 2.2) are divided into connection and output equations. It is shown that the connection defect can be calculated (Theorem 5.2) and the output defect estimated (Theorem 5.9).

Contribution 3. An automated configuration method for all stages of model-based development by evaluating co-simulation quality.

The method for the automated configuration of a DEW (Definition 5.6) is presented in Algorithm 5.1. This method finds the number of initial tokens, the initial token values and the simulator step sizes. The same parameters can be used later to configure the CSW (Definition 3.25). Example 5.15 shows the application of the introduced algorithm for configuring the model of computation.

The simulator step sizes are adjusted according to the method shown in Definition 5.14. Theorem 2.17 and Theorem 5.4 show that both connection and output errors can be controlled by reducing the simulator step sizes. The integration defect is controlled by internal solvers contained in the CSLs (Figure 1.1). Such a distribution of responsibility is consistent with the black box nature of CSLs (Figure 5.1). For CSLs with low integration defects and low requested tolerance, Theorem 2.12 guarantees that the co-simulation error should be low. Algorithm 5.1 iteratively adjusts the simulator step sizes until the requirement tolerance is met.

In each iteration of the algorithm, the number of initial tokens and initial token values are calculated. The method introduced in Definition 4.8 is used to determine the number of tokens given the simulator step sizes. This method ensures that the resulting DEW does not deadlock (Corollary 4.13). In addition, it can be estimated whether the DEW can be executed in real time (Theorem 4.17). An optimization approach is used to determine initial token values (Definition 5.11).

To the best of the author's knowledge, Contribution 3 is an original contribution of this thesis. Algorithm 5.1 is similar to the algorithm presented in [43] and was developed independently. The advantage of Algorithm 5.1 is that theoretical analysis is based on CODESYS (Definition 2.2). The co-simulation quality criterion does not have to have a physical meaning such as energy residual. In addition, Algorithm 5.1 tunes the co-simulation with multiple rates and results in a determinate MOC.

6.2 Future research

A CODESYS (Definition 2.2) is the basis for the numerical error analysis in this work. Part of future research is to examine how well hybrid behavior [62] can be approximated with a CSW. Existing reports on the use of co-simulation in hybrid electrical vehicle design [79, 80, 81, 82, 83, 84] suggest that such an analysis can be of practical benefit.

Rate converters are defined in this thesis as zero order holding elements (Definition 3.24). There are coupling elements like those introduced in [53, 68, 69]. The CSW can be generalized to rate converters with several inputs and outputs. The notation used to describe such rate converters is more complex. This can lead to a more complex formulation of the results in this thesis such as Theorem 3.29. The work in this thesis already enables an objective comparison between different coupling elements for particular (Definition 5.10). It is interesting to examine whether a guideline can be developed on the basis of this criterion.

Theorem 4.17 assumes a parallel execution on an IPEP (Definition 4.1). It shows that whether given enough processors and simulators capable of running in real-time the CSW can be executed in real-time. There is an implicit trade-off between the quality of the simulation and the real-time capabilities, as the parallel simulation tends to be less stable than the sequential one [31]. In future work it is planned to enable a more complex description of the platform, e.g. finite processors or different processor speeds [70]. The goal is to analyze if the heuristics proposed for the number of initial tokens (Definition 4.8) can be improved. Existing research results for scheduling SDFGs [36] can be used to improve the method.

The numerical stability is not analyzed in this thesis. The zero stability is analyzed in [7]. The stability analysis is difficult for the co-simulation, since even the zero stability depends on the existence of algebraic loops. Relative stability is only analyzed on test models such as two-mass oscillators [15, 31, 54]. It remains to be investigated how a relative stability measure can be formulated for a co-simulation with a larger number of slaves and ports. This should be the most interesting topic in future work. If such a measure exists, there may be an opportunity to develop an automated configuration algorithm that does not require co-simulation runs.

Bibliography

[1]C. Steinbrink, S. Lehnhoff, S. Rohjans, T. I. Strasser, E. Widl, C. Moyo, G. Lauss, F. Lehfuss, M. Faschang, P. Palensky *et al.*, “Simulation-based validation of smart grids–status quo and future research trends,” in *International Conference on Industrial Applications of Holonic and Multi-Agent Systems*. Springer, 2017, pp. 171–185, https://doi.org/10.1007/978-3-319-64635-0_13.

[2]C. Gomes, C. Thule, D. Broman, P. G. Larsen, and H. Vangheluwe, “Co-simulation: A survey,” *ACM Comput. Surv.*, vol. 51, no. 3, pp. 49:1–49:33, May 2018, http://doi.org/10.1145/3179993.

[3]T. Blochwitz, M. Otter, M. Arnold, C. Bausch, H. Elmqvist, A. Junghanns, J. Mauß, M. Monteiro, T. Neidhold, D. Neumerkel *et al.*, “The functional mockup interface for tool independent exchange of simulation models,” in *Proceedings of the 8th International Modelica Conference; March 20th-22nd; Technical Univeristy; Dresden; Germany*. Linköping University Electronic Press, 2011, pp. 105–114.

[4]T. Blochwitz, M. Otter, J. Akesson, M. Arnold, C. Clauss, H. Elmqvist, M. Friedrich, A. Junghanns, J. Mauss, D. Neumerkel *et al.*, “Functional mockup interface 2.0: The standard for tool independent exchange of simulation models,” in *Proceedings of the 9th International MODELICA Conference; September 3-5; 2012; Munich; Germany*. Linköping University Electronic Press, 2012, pp. 173–184, https://doi.org/10.3384/ecp12076173.

[5]“Functional Mock-up Interface for Model Exchange and Co-Simulation, Version 2.0,” https://svn.modelica.org/fmi/branches/public/specifications/v2.0/FMI_for_ModelExchange_and_CoSimulation_v2.0.pdf, 2014, accessed 30 August 2019.

[6]“Tools | Functional Mock-up Interface,” https://fmi-standard.org/tools/, accessed 8 June 2020.

[7]R. Kübler and W. Schiehlen, “Two Methods of Simulator Coupling,” *Mathematical and Computer Modelling of Dynamical Systems*, vol. 6, no. 2, pp. 93–113, Jun. 2000, http://doi.org/10.1076/1387-3954(200006)6:2;1-M;FT093.

[8]B. P. Zeigler, T. G. Kim, and H. Praehofer, *Theory of modeling and simulation.* Academic press, 2000.

[9]F. S. Merritt and J. T. Ricketts, *Building design and construction handbook.* Citeseer, 2001, vol. 13.

[10]R. Isermann, J. Schaffnit, and S. Sinsel, "Hardware-in-the-loop simulation for the design and testing of engine-control systems," *Control Engineering Practice*, vol. 7, no. 5, pp. 643–653, 1999, https://doi.org/10.1016/S0967-0661(98)00205-6.

[11]P. J. Mosterman, S. Prabhu, and T. Erkkinen, "An industrial embedded control system design process," *Proceedings of the Canadian Engineering Education Association (CEEA)*, 2004, https://doi.org/10.24908/pceea.v0i0.4005.

[12]E. Bringmann and A. Krämer, "Model-based testing of automotive systems," in *2008 1st international conference on software testing, verification, and validation.* IEEE, 2008, pp. 485–493, https://doi.org/10.1109/ICST.2008.45.

[13]J. F. Broenink and Y. Ni, "Model-driven robot-software design using integrated models and co-simulation," in *2012 International Conference on Embedded Computer Systems (SAMOS).* IEEE, 2012, pp. 339–344, https://doi.org/10.1109/SAMOS.2012.6404197.

[14]E. Drenth, M. Törmänen, K. Johansson, B.-A. Andersson, D. Andersson, I. Torstensson, and J. Åkesson, "Consistent simulation environment with fmi based tool chain," in *Proceedings of the 10 th International Modelica Conference; March 10-12; 2014; Lund; Sweden.* Linköping University Electronic Press, 2014, pp. 1277–1283, https://doi.org/10.3384/ecp140961277.

[15]B. Schweizer, P. Li, and D. Lu, "Explicit and implicit cosimulation methods: Stability and convergence analysis for different solver coupling approaches," *Journal of Computational and Nonlinear Dynamics*, vol. 10, no. 5, p. 051007, 2015, https://doi.org/10.1115/1.4028503.

[16]H.-E. Eriksson and M. Penker, *Business modeling with UML: business patterns at work.* John Wiley & Sons, 2000.

[17]A. Kossiakoff, W. N. Sweet, S. J. Seymour, and S. M. Biemer, *Systems engineering principles and practice.* John Wiley & Sons, 2011, vol. 83.

[18]B. Selic, "The pragmatics of model-driven development," *IEEE Software*, vol. 20, no. 5, pp. 19–25, Sep. 2003, http://doi.org/10.1109/MS.2003.1231146.

[19]M. Broy, S. Kirstan, H. Krcmar, and B. Schätz, “What is the benefit of a model-based design of embedded software systems in the car industry?” in *Emerging Technologies for the Evolution and Maintenance of Software Models*. IGI Global, 2012, pp. 343–369, http://doi.org/10.4018/978-1-61350-438-3.ch013.

[20]A. M. Madni and S. Purohit, “Economic analysis of model-based systems engineering,” *Systems*, vol. 7, no. 1, p. 12, 2019, http://doi.org/10.3390/systems7010012.

[21]E. A. Lee, “Cyber physical systems: Design challenges,” in *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*, May 2008, pp. 363–369, http://doi.org/10.1109/ISORC.2008.25.

[22]P. Derler, E. A. Lee, and A. S. Vincentelli, “Modeling cyber–physical systems,” *Proceedings of the IEEE*, vol. 100, no. 1, pp. 13–28, 2011, http://doi.org/10.1109/JPROC.2011.2160929.

[23]E. Lee, “The past, present and future of cyber-physical systems: A focus on models,” *Sensors*, vol. 15, no. 3, pp. 4837–4869, 2015, http://doi.org/10.3390/s150304837.

[24]E. A. Lee, “Disciplined heterogeneous modeling,” in *Model Driven Engineering Languages and Systems*, D. C. Petriu, N. Rouquette, and Ø. Haugen, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 273–287, http://doi.org/10.1007/978-3-642-16129-2_20.

[25]P. A. Emrath and D. A. Padua, “Automatic detection of nondeterminacy in parallel programs,” in *Proceedings of the 1988 ACM SIGPLAN and SIGOPS workshop on Parallel and distributed debugging*, 1988, pp. 89–99, https://doi.org/10.1145/69215.69224.

[26]R. H. Netzer and B. P. Miller, “What are race conditions? some issues and formalizations,” *ACM Letters on Programming Languages and Systems (LOPLAS)*, vol. 1, no. 1, pp. 74–88, 1992, https://doi.org/10.1145/130616.130623.

[27]K. Zandberg, “Dataflow-based model-driven engineering of control systems,” Master’s thesis, University of Twente, 2020, http://purl.utwente.nl/essays/80707.

[28]E. A. Lee and D. G. Messerschmitt, “Synchronous data flow,” *Proceedings of the IEEE*, vol. 75, no. 9, pp. 1235–1245, Sep. 1987, http://doi.org/10.1109/PROC.1987.13876.

[29]D. Broman, C. Brooks, L. Greenberg, E. A. Lee, M. Masin, S. Tripakis, and M. Wetter, “Determinate composition of fmus for co-simulation,” in *Proceedings of the Eleventh ACM International Conference on Embedded Software*. IEEE, 2013, p. 2, https://doi.org/10.1109/EMSOFT.2013.6658580.

[30]M. Arnold and M. Günther, “Preconditioned dynamic iteration for coupled differential-algebraic systems,” *BIT Numerical Mathematics*, vol. 41, no. 1, pp. 1–25, Jan 2001, http://doi.org/10.1023/A:1021909032551.

[31]M. Busch, “Continuous approximation techniques for co-simulation methods: Analysis of numerical stability and local error,” *ZAMM - Journal of Applied Mathematics and Mechanics*, vol. 96, no. 9, pp. 1061–1081, 2016, https://doi.org/10.1002/zamm.201500196.

[32]C. W. Gear and D. Wells, “Multirate linear multistep methods,” *BIT Numerical Mathematics*, vol. 24, no. 4, pp. 484–502, 1984.

[33]B. Thiele, M. Otter, and S. E. Mattsson, “Modular multi-rate and multi-method real-time simulation,” in 10^{th} *International Modelica Conference*, 2014, pp. 381–393, https://doi.org/10.3384/ecp14096381.

[34]R. M. Karp and R. E. Miller, “Properties of a model for parallel computations: Determinacy, termination, queueing,” *SIAM Journal on Applied Mathematics*, vol. 14, no. 6, pp. 1390–1411, 1966.

[35]E. A. Lee and D. G. Messerschmitt, “Static scheduling of synchronous data flow programs for digital signal processing,” *IEEE Transactions on Computers*, vol. C-36, no. 1, pp. 24–35, Jan 1987.

[36]S. Sriram and S. S. Bhattacharyya, *Embedded multiprocessors: Scheduling and Synchronization*, 2nd ed. CRC press, 2009, ISBN: 978-1-4200-4801-8.

[37]S. Tripakis, “Bridging the semantic gap between heterogeneous modeling formalisms and fmi,” in *2015 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*, July 2015, pp. 60–69, http://doi.org/10.1109/SAMOS.2015.7363660.

[38]L. F. Shampine, “Error estimation and control for odes,” *Journal of Scientific Computing*, vol. 25, no. 1-2, pp. 3–16, 2005.

[39]M. Busch and B. Schweizer, “An explicit approach for controlling the macro-step size of co-simulation methods,” in *Proceedings of the 7th European Nonlinear Dynamics Conference (ENOC 2011): July 24 - 29, 2011, Rome, Italy*, D. Bernadini, Ed. ., 2011, pp. 1–6. [Online]. Available: http://tubiblio.ulb.tu-darmstadt.de/77923/

[40]M. Arnold, C. Clauß, and T. Schierz, “Error analysis and error estimates for co-simulation in fmi for model exchange and co-simulation v2. 0,” in *Progress in Differential-Algebraic Equations*. Springer, 2014, pp. 107–125, 10.2478/meceng-2013-0005.

[41]P. H. Nguyen, *Interpolation and error control schemes for algebraic differential equations using continuous implicit Runge-Kutta methods*. University of Toronto, Department of Computer Science, 1995.

[42]M. Benedikt and F. R. Holzinger, "Automated configuration for non-iterative co-simulation," in *2016 17th International Conference on Thermal, Mechanical and Multi-Physics Simulation and Experiments in Microelectronics and Microsystems (EuroSimE)*, April 2016, pp. 1–7, http://doi.org/10.1109/EuroSimE.2016.7463355.

[43]J. Rahikainen, F. González, and M. Á. Naya, "An automated methodology to select functional co-simulation configurations," *Multibody System Dynamics*, vol. 48, no. 1, pp. 79–103, Jan 2020, http://doi.org/10.1007/s11044-019-09696-y.

[44]"Github - sglumac/sdf4sim," https://github.com/sglumac/sdf4sim, 2020, accessed 14 October 2020.

[45]"Model | Definition of model at Dictionary.com," https://dictionary.cambridge.org/dictionary/english/model, accessed 7 June 2020.

[46]E. Hairer and G. Wanner, *Solving ordinary differential equations II, Stiff and Differential-Algebraic Problems*. Springer Berlin Heidelberg, 1996.

[47]W. Enright, "Continuous numerical methods for odes with defect control," *Journal of Computational and Applied Mathematics*, vol. 125, no. 1-2, pp. 159–170, 2000.

[48]"Explore windows 10 os, computers, apps, & more | microsoft," https://www.microsoft.com/en-us/windows, accessed 8 June 2020.

[49]"Tenasys intime for windows - consolidate hard rtos with windows," https://www.tenasys.com/products/intime-rtos/intime-for-windows/, accessed 8 June 2020.

[50]H. Ernst, N. S. Paul, and W. Gerhard, *Solving ordinary differential equations I: Nonstiff problems*. Berlin, New York: Springer-Verlag, 2008.

[51]M. Wetter, "Co-simulation of building energy and control systems with the building controls virtual test bed," *Journal of Building Performance Simulation*, vol. 4, no. 3, pp. 185–203, 2011, https://doi.org/10.1080/19401493.2010.518631.

[52]N. Pedersen, T. Bojsen, J. Madsen, and M. Vejlgaard-Laursen, "FMI for co-simulation of embedded control software," in *1st Japanese Modelica Conference 2016*. Linköping University Electronic Press, 2016, pp. 70–77, http://doi.org/10.3384/ecp1612470.

[53]M. Benedikt, D. Watzenig, and A. Hofer, “Modelling and analysis of the non-iterative coupling process for co-simulation,” *Mathematical and computer modelling of dynamical systems*, vol. 19, no. 5, pp. 451–470, 2013, https://doi.org/10.1080/13873954.2013.784340.

[54]B. Schweizer and D. Lu, “Predictor/corrector co-simulation approaches for solver coupling with algebraic constraints,” *ZAMM-Journal of Applied Mathematics and Mechanics/Zeitschrift für Angewandte Mathematik und Mechanik*, vol. 95, no. 9, pp. 911–938, 2015.

[55]S. Glumac and Z. Kovacic, “Relative consistency and robust stability measures for sequential co-simulation,” in *Proceedings of the 13th International Modelica Conference, Regensburg, Germany, March 4–6, 2019*, no. 157. Linköping University Electronic Press, 2019, https://doi.org/10.3384/ecp19157197.

[56]O. de Oliveira, “The implicit and the inverse function theorems: Easy proofs,” *Real Analysis Exchange*, vol. 39, no. 1, pp. 207–218, 2014, https://doi.org/10.14321/realanalexch.39.1.0207.

[57]E. Süli and D. F. Mayers, *An introduction to numerical analysis*. Cambridge university press, 2003.

[58]R. C. Dorf and R. H. Bishop, *Modern Control Systems*, 12th ed. Pearson, 2011, ISBN: 978-0-13-602458-3.

[59]W. Borutzky, *Bond graph modelling of engineering systems*. Springer, 2011, vol. 103.

[60]S. S. Dragomir, *Some Gronwall type inequalities and applications*. Nova Science Publishers New York, 2003.

[61]L. Shampine, “Solving odes and ddes with residual control,” *Applied Numerical Mathematics*, vol. 52, no. 1, pp. 113–127, 2005, https://doi.org/10.1016/j.apnum.2004.07.003.

[62]D. Broman, L. Greenberg, E. A. Lee, M. Masin, S. Tripakis, and M. Wetter, “Requirements for hybrid cosimulation standards,” in *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control*, ser. HSCC ’15. New York, NY, USA: ACM, 2015, pp. 179–188, https://doi.org/10.1145/2728606.2728629.

[63]Z. Jackiewicz, *General linear methods for ordinary differential equations*. Wiley Online Library, 2009.

[64]A. Jantsch, *Modeling Embedded Systems and SoC’s: Concurrency and Time in Models of Computation*. Elsevier, 2003.

[65]E. A. Lee and S. Ha, “Scheduling strategies for multiprocessor real-time dsp,” in *1989 IEEE Global Telecommunications Conference and Exhibition ’Communications Technology for the 1990s and Beyond’*, 1989, pp. 1279–1283 vol.2, https://doi.org/10.1109/GLOCOM.1989.64160.

[66]S. E. Saidi, N. Pernet, and Y. Sorel, “A method for parallel scheduling of multi-rate co-simulation on multi-core platforms,” *Oil & Gas Science and Technology–Revue d’IFP Energies nouvelles*, vol. 74, p. 49, 2019, https://doi.org/10.2516/ogst/2019009.

[67]G. H. Hardy, E. M. Wright *et al.*, *An introduction to the theory of numbers*. Oxford university press, 1979.

[68]M. Benedikt, D. Watzenig, J. Zehetner, and A. Hofer, “Nepce-a nearly energy-preserving coupling element for weak-coupled problems and co-simulation,” in *International Conference on Computational Methods for Coupled Problems in Science and Engineering*. ., 2013, pp. 1–12.

[69]G. Stettinger, M. Horn, M. Benedikt, and J. Zehetner, *A model-based approach for prediction-based interconnection of dynamic systems*. ., 2014, pp. 3286–3291.

[70]M. Pinedo, *Scheduling*. Springer, 2012, vol. 5.

[71]J. Y. Leung, *Handbook of scheduling: algorithms, models, and performance analysis*. CRC press, 2004.

[72]J. E. Kelley Jr, “Critical-path planning and scheduling: Mathematical basis,” *Operations research*, vol. 9, no. 3, pp. 296–320, 1961.

[73]J. L. Pino, S. S. Bhattacharyya, and E. A. Lee, *A hierarchical multiprocessor scheduling framework for synchronous dataflow graphs*. Electronics Research Laboratory, College of Engineering, University of California, 1995, https://ptolemy.berkeley.edu/publications/papers/95/erl-95-36/.

[74]F. Baccelli, G. Cohen, G. J. Olsder, and J.-P. Quadrat, “Synchronization and linearity: an algebra for discrete event systems,” 1992.

[75]K. Beck, *Test-driven development: by example*. Addison-Wesley Professional, 2003.

[76]A. Spitzbart, “A generalization of hermite’s interpolation formula,” *The American Mathematical Monthly*, vol. 67, no. 1, pp. 42–46, 1960, https://doi.org/10.2307/2308924.

[77]F. Gao and L. Han, “Implementing the nelder-mead simplex algorithm with adaptive parameters,” *Computational Optimization and Applications*, vol. 51, no. 1, pp. 259–277, 2012, https://doi.org/10.1007/s10589-010-9329-3.

[78]“minimize(method=’Nelder-Mead’) SciPy v1.5.2 Reference Guide,” https://docs.scipy.org/doc/scipy/reference/optimize.minimize-neldermead.html, 2020, accessed 20 September 2020.

[79]B. Cho and N. Vaughan, “Dynamic simulation model of a hybrid powertrain and controller using co-simulation - part ii: Control strategy,” *International journal of Automotive technology*, vol. 7, no. 7, pp. 785–793, 2006.

[80]J. Hong, S. Kim, and B. Min, “Drivability development based on cosimulation of amesim vehicle model and simulink hcu model for parallel hybrid electric vehicle,” SAE Technical Paper, Tech. Rep., 04 2009, https://doi.org/10.4271/2009-01-0725.

[81]D.-x. Zhang, X.-h. Zeng, P.-y. Wang, and Q.-n. Wang, “Co-simulation with amesim and matlab for differential dynamic coupling of hybrid electric vehicle,” in *2009 IEEE Intelligent Vehicles Symposium.* IEEE, 2009, pp. 761–765, https://doi.org/10.1109/IVS.2009.5164373.

[82]Q. Zhang, N. Cui, K. Li, Y. Shang, and C. Zhang, “Co-simulation of energy management strategy for hybrid electric vehicle in avl inmotion,” in *2017 Chinese Automation Congress (CAC)*, 2017, pp. 4932–4937, https://doi.org/10.1109/CAC.2017.8243653.

[83]J. J. Eckert, F. M. Santiciolli, L. C. Silva, E. S. Costa, F. C. Corrêa, and F. G. Dedini, “Co-simulation to evaluate acceleration performance and fuel consumption of hybrid vehicles,” *Journal of the Brazilian Society of Mechanical Sciences and Engineering*, vol. 39, no. 1, pp. 53–66, 2017, https://doi.org/10.1007/s40430-015-0484-4.

[84]R. Yuan, T. Fletcher, A. Ahmedov, N. Kalantzis, A. Pezouvanis, N. Dutta, A. Watson, and K. Ebrahimi, “Modelling and co-simulation of hybrid vehicles: A thermal management perspective,” *Applied Thermal Engineering*, p. 115883, 2020, https://doi.org/10.1016/j.applthermaleng.2020.115883.

Glossary

Acronyms

CODE coupled ordinary differential equations 9, 10

CODESUB CODE subsystem 11, 12, 14, 15, 24–27, 50, 79, 86, 88–91, 96, 115–125

CODESYS CODE system . 6, 9–11, 13–15, 17, 18, 20, 23, 25–27, 84, 86, 87, 94, 95, 98, 102, 103, 119, 121, 122, 125

CSL co-simulation slave 1, 4, 6, 7, 10, 20–25, 30, 50–55, 58, 83, 89, 90, 98, 100–102, 117–123, 125

CSM co-simulation master 1–3, 5–7, 23, 26, 27, 29, 45, 50, 60, 83

CSN co-simulation network . . 2, 3, 6, 22, 23, 25, 29, 50, 53, 55, 56, 58, 59, 66, 78, 79, 90, 93, 98, 101, 114, 115, 119, 122

CSW co-simulation network wrapper 29, 50, 52–62, 66, 67, 71, 73, 74, 77–80, 83, 84, 86–88, 92–95, 97, 98, 101–103, 118, 119

DAE differential-algebraic equation 10, 25

DEW defect estimation wrapper 89, 90, 92, 93, 96–98, 103, 114, 117, 119

FMI Functional Mock-up Interface . . . . xi, xiii, 1–4, 9, 10, 20–22, 50

FMU Functional Mock-up Unit 1, 2, 5, 10, 20, 21, 29, 79, 101

HIL hardware in the loop 2, 4, 7, 10, 25, 61, 62, 79, 80, 82, 92

IPEP infinite processor execution platform 6, 61–63, 65, 66, 74, 78, 103, 122

MBD model-based development 2–4, 7, 25, 62, 78–82

MIL model in the loop 2, 4, 78–82, 92

MOC model of computation 2, 3, 5, 6, 29, 38, 44, 45, 49, 50, 54, 60–62, 103

ODE ordinary differential equation 2, 10, 13, 25

ODEST output defect estimator 89–91, 93, 96, 97, 102, 115–117, 121–125

PASS periodic adimissible sequential schedule 45, 47–50, 63, 67, 69, 71, 115, 121

ROL rest of the loop 3, 6, 78

SDF synchronous data flow 3–6, 29, 46, 50, 61, 82

SDFA SDF actor 30–33, 35–38, 45–47, 50–57, 59, 62, 63, 65–67, 74, 78–81, 85, 89, 114, 115, 117–121, 123–125

SDFB SDF buffer 31, 34–38, 41, 45, 46, 54, 55, 59, 62, 67–72, 74, 76–79, 115, 117–120

SDFG SDF graph ix, 5–7, 29–32, 34, 35, 37–39, 42–50, 53, 59, 61–66, 73, 78–80, 82, 90, 92, 93, 101, 103, 115, 117–122

SDFI SDF iteration 50, 58, 61–63, 65, 66, 77, 97, 118, 122

SIL software in the loop 79–82

SUBW CODE subsystem wrapper 23–28, 90, 91, 115, 116, 122–125

SYSW CODE system wrapper 20, 23, 25–28, 79, 87, 94, 96, 98, 99

UML Unified Modeling Language 2

XIL X in the loop 4, 78, 79

ZOH zero-order hold 22, 26, 53, 59, 81, 94

Symbols

A an SDFA 30, 51, 52, 66, 85, 86, 89–91

- A_a the SDFA with label a .....
  - A_1 ..... 32, 33, 49, 80, 81, 94
  - A_2 ..... 32, 33, 49, 94
  - A_3 94
  - $A_{\underline{\mathrm{a}}}$ the rate converter represented as the $\underline{\mathrm{a}}^{th}$ SDFA 85
  - $A_{\breve{\imath}}$ 85
  - $A_{\hat{\imath}}$ 85
- A' an SDFA 89

adaptStepSizes the function used to adjust the step sizes for the DEW 97, 98

analytic an analytic solver 27, 28

A the function labeling SDFAs 31, 32, 53, 54, 65, 66, 75, 80, 81, 85, 90

- A' the function labeling SDFAs in a DEW 90

$\mathcal{A}$ the set of all possible SDFAs 31, 32, 51, 52, 62, 89

C a CSN 22, 25, 27, 28, 53, 55, 56, 58, 59, 66, 67, 78–80, 84, 86, 87, 90, 93, 94, 96–98

c the function that assigns a consumption rate to the input port in an SDFA 30, 31, 36, 51, 52, 89

- c_a ... in the SDFA labeled a 36, 37, 39, 44, 46, 47, 57, 63, 64, 74, 75
  - c_1 32
  - c_2 33
  - c_{a_0} 40
  - $\mathrm{c}_{a''}$ 75
  - $\mathrm{c}_{\breve{a}}$ 38, 39, 41–43, 64, 74–77
  - $\mathrm{c}_{\underline{\mathrm{a}}}$ 56, 76, 77, 85, 86

$c_{a'}$ 63, 70, 71

$c_{\breve{\imath}}$ 56

calculate the calculation function of an SDFA 30, 51, 52, 89

calculate$_a$ . . . of the SDFA labeled a 36, 44, 45, 47, 62

calculate$_1$ 32, 33

calculate$_2$ 33

calculate$'$ 89

choose the function that chooses an SDFA when searching for a PASS 47–49, 67

consume the function that carries out the consumption of tokens 35–43, 45, 47, 62, 64

converter the function used to obtain a rate converter for the connection between two simulators 52, 54

count the function used to count the number of operations during an SDFG execution . 35, 64

$\mathcal{C}$ the set of all possible CSNs 53, 66, 98

d the number of tokens in the SDFBs 35–39, 42, 45, 47, 62, 63, 67–73

d_0 the number of initial tokens in the SDFBs 31, 34, 35, 38–43, 45, 47, 49, 53, 54, 56, 58–60, 63, 64, 67, 71–80, 84, 86, 87, 90, 93, 94, 96–98

d_{01} 65, 94

d_{02} 66, 94, 95

d_{03} 94

δA the defect of an ODEST 96, 97

$\hat{\boldsymbol{\delta}}\widetilde{\mathbf{Y}}'$ the defect of the the numerical solution found for the output signal of all CODESUBs 93

$\hat{\boldsymbol{\delta}}\widetilde{\mathbf{y}}'$ the estimate of the output defect of the ODEST 91, 92

$\delta^{(t_{\kappa-1},t]}$ the maximum defect on the interval $(t_{\kappa-1},t]$ 19, 20

$\delta^{(t_0,t]}$ the maximum defect on the interval $(t_0,t]$ 19, 20

$\Delta\widetilde{\mathbf{U}}$ the global error of the numerical solution found for the input signal of all CODESUBs 18, 19

$\delta\widetilde{\mathbf{U}}$ the defect of the numerical solution found for the input signal of all CODESUBs . 18–20, 83

$\delta\widetilde{\mathbf{U}}'$ the defect of the numerical solution found for the input signal of all CODESUBs 93

$\delta\widetilde{u}_{i\iota}$ the defect of the numerical solution for the signal consumed by the ι^{th} port of the i^{th} CODESUB 96

$\delta\widetilde{u}_{\breve{\imath}\breve{\iota}}$ 6, 17, 18, 86, 88, 96

$\delta\underline{\mathbf{x}}$ the defect of the state signal of a SUBW 27, 28

$\delta\underline{\mathbf{x}}_i$ . . . of the SUBW labeled i 26

$\Delta\widetilde{\mathbf{X}}$ the global error of the numerical solution found for the state signal of all CODESUBs 18–20

$\delta\widetilde{\mathbf{X}}$ the defect of the numerical solution found for the state signal of all CODESUBs 19

$\delta\widetilde{\mathbf{x}}$ the defect of the state signal of a CODESUB 87

- $\delta\widetilde{\mathbf{x}}_i$ ... of the i^{th} CODESUB 6, 17, 18, 88
  - $\delta\widetilde{\mathbf{x}}_1$ 83
  - $\delta\widetilde{\mathbf{x}}_2$ 83

$\delta\underline{\mathbf{y}}$ the defect of the output signal of a SUBW 26

- $\delta\underline{\mathbf{y}}_i$ ... of the SUBW labeled i 26

$\Delta\widetilde{\mathbf{Y}}$ the global error of the the numerical solution found for the output signal of all CODESUBs 18, 20

$\delta\widetilde{\mathbf{Y}}$ the defect of the the numerical solution found for the output signal of all CODESUBs 18–20

$\delta\widetilde{\mathbf{y}}$ the defect of the output signal of a CODESUB ....

- $\delta\widetilde{\mathbf{y}}_i$ the defect of the output signal of the i^{th} CODESUB 6, 17, 18
  - $\delta\widetilde{\mathbf{y}}_1$ 83
  - $\delta\widetilde{\mathbf{y}}_2$ 83

$\delta\widetilde{\mathbf{y}}'$ the output defect of the ODEST 91, 92

$\Delta\widetilde{y}_{io}$ the global error of the the numerical solution found for the output signal of the i^{th} CODESUBs and its output labeled o ....

- $\Delta\widetilde{y}_{21}$ 96
- $\Delta\widetilde{y}_{31}$ 96

$\delta\widetilde{y}_{io}$ the defect of the numerical solution for the signal produced by the o^{th} port of the i^{th} CODESUB 96

d_i the denominator of the step size assigned to the i^{th} simulator ....

- $d_{\breve{\imath}}$ the denominator of the step size assigned to the $\breve{\imath}^{th}$ simulator 54
- $d_{\hat{\imath}}$ the denominator of the step size assigned to the $\hat{\imath}^{th}$ simulator 54

doStep the step function of an co-simulation slave 21, 22, 24, 25, 51, 89, 90

dst the destination port consuming the tokens from the buffer 31, 34, 36–41, 43, 46, 47, 53–56, 63–74, 76, 90, 95

E a co-simulation experiment 78–81

η_i the numerator of the step size assigned to the i^{th} simulator ....

- $\eta_{\breve{\imath}}$ the numerator of the step size assigned to the $\breve{\imath}^{th}$ simulator 54
- $\eta_{\hat{\imath}}$ the numerator of the step size assigned to the $\hat{\imath}^{th}$ simulator 54

euler a forward Euler solver 27

F a co-simulation slave 21, 24, 26, 51, 86, 89–91

- F_i the co-simulation slave with label i ....
  - $F_{\breve{\imath}}$ 53
  - $F_{\hat{\imath}}$ 53

$\mathbf{F}$ the aggregation of state transition functions of all CODESUBs 12, 13, 19

f the state transition function of a CODESUB . . . 11, 27, 28, 86, 87

  $\mathbf{f}_i$ . . . of the i^{th} CODESUB . . . 6, 11, 17, 26, 88

    $\mathbf{f}_1$ . . . 14, 16, 83

    $\mathbf{f}_2$ . . . 14, 16, 83

    $\mathbf{f}_3$ . . . 16

findTokenNumbers the function that generates the number of initial tokens on each SDFB 67, 98

findTokenValues the function that generates the values of initial tokens . . . 93, 94, 98

F the function labeling CSLs . . . 22, 25, 54, 90

$\mathcal{F}$ the set of all possible CSLs . . . 22, 24, 51, 89

G an SDFG 31, 45, 47, 50, 53, 56, 58, 59, 62, 63, 65, 67, 74–78, 80, 81, 84, 86, 87, 90, 96–98

  G_1 . . . 65, 66, 94

  G_2 . . . 66, 94

  G_3 . . . 94

  G' a DEW . . . 78, 79, 90

γ the function referencing each individual output token produced by an SDFA . . . 30, 52

  γ_a . . . by the SDFA labeled a . . . 36, 37, 44, 45, 47, 62

    γ_1 . . . 33

    γ_2 . . . 33, 96

    γ_3 . . . 96

    $\gamma_{\hat{a}}$ . . . 41–43

    $\gamma_{\underline{a}}$ . . . 85

    γ'_a . . . 44

    γ''_a . . . 44

    $\gamma_{\hat{\imath}}$ . . . 84–86

$\mathbf{\Gamma}$ the topology matrix of an SDF graph . . . 46–48, 56, 57

γ' . . . by an ODEST . . . 90

G the aggregation of output functions of all CODESUBs . . . 12, 13, 18, 20

g the output function of a CODESUB . . . 11, 26, 91, 92

  $\mathbf{g}_i$ . . . of the i^{th} CODESUB . . . 6, 11, 17, 26, 88, 91

    $\mathbf{g}_1$ . . . 14, 16, 83

    $\mathbf{g}_2$ . . . 14, 16, 83

    $\mathbf{g}_3$ . . . 16

  $\mathbf{g}_{\hat{\imath}}$ . . . 88

gcd greatest common divisor . . . 46, 47, 54, 56–58

get the get function of an co-simulation slave . . . 21, 22, 24, 25, 52, 89, 90

g_o the output function generating a signal for the o^{th} output port of a CODESUB . . . 25

$\mathcal{G}$ the set of all possible SDFGs 45, 50, 53, 62, 63, 65, 78, 97

H the simulated time increment in a single iteration of the CSW 58, 75–77, 88, 93, 94

h the step size assigned to the simulator 51, 67, 78, 80, 86, 87, 89–92

$\mathbf{H}$ the function used in an explicit expression for the analytical input signals 13, 18

$\widetilde{\mathbf{H}}$ the function used in an explicit expression for the numerical input signals 18, 19

$\boldsymbol{h}$ a sequence of communication step sizes assigned to a CSL 21, 25–27, 91

h_{init} the initial step size in the automatic configuration algorithm 98

h' the function selecting the generalized step sizes of SDFAs in a CSW 55–57

h the function selecting the simulator step sizes . 53, 54, 56, 58, 59, 66, 74, 75, 77, 79, 84–88, 90, 93, 96–98

h_1 94

h_2 94

h_3 . . . . 94, 95

h' 97

I^{A} the set of SDFA labels . 31, 32, 34–36, 38–40, 44–47, 50, 53, 55, 56, 58, 62, 63, 65, 66, 90

I^{B} the set of SDFB labels . 31, 34, 35, 38, 41, 47, 53, 54, 56, 62–66, 78, 79, 85, 90, 93–95, 97

$\mathbf{I}$ identity matrix 13, 18

I_{F} the set of CSL labels 22, 25, 53–56, 66, 67, 88, 90, 93–95, 97, 98

I_{M} the set of CODESUB labels 11, 14, 15, 25, 88

iteration an SDFI 50, 65, 74, 75

I_{U} the set of input port label sets used in a CSL 21, 22, 24, 51

I_{U_i} . . . in the CSL labeled i 54, 55, 67, 95

I_{U_a} 74, 75

$I_{\mathrm{U}_{\hat{\imath}}}$ 54, 56

$I_{\mathscr{U}}$ the set of input port labels used in an SDFA 30, 51, 52, 89

$I_{\mathscr{U}_a}$ . . . in the SDFA labeled a 36, 39, 44, 47, 63, 67–69, 72, 73

$I_{\mathscr{U}_1}$ 32, 33

$I_{\mathscr{U}_2}$ 33

$I_{\mathscr{U}_{a_0}}$ 40

$I_{\mathscr{U}_{\check{a}}}$ 38, 64

$I_{\mathscr{U}_{a'}}$ 63, 68, 69

$I_{\mathscr{U}_{\hat{\imath}}}$ 85

I_{Y} the set of output port label sets used in a CSL 21, 22, 24, 51, 89, 90

I_{Y_i} . . . in the CSL labeled i . . . .

$I_{\mathrm{Y}_{\hat{\imath}}}$ 56

$I_{\mathscr{Y}}$ the set of output port labels used in an SDFA 21, 30, 51, 52, 89

$I_{\mathscr{Y}_a}$ . . . in the SDFA labeled a 67–69, 72, 73

$I_{\mathscr{Y}_1}$ 32, 33

$I_{\mathscr{Y}_2}$ 33

$I_{\mathscr{Y}_{\hat{a}}}$ 38, 63, 64

$I_{\mathscr{Y}_{a'}}$ 68, 69

$I_{\mathscr{Y}_{\hat{i}}}$ 85

$I_{\mathscr{Y}'}$ 89

J the criterion for the quality assessment of a DEW 93, 97, 98

$K_{\mathbf{F}}$ the Lipchitz constant of the function $\mathbf{f}$ 12, 13, 19, 20

$K_{\mathbf{f}}$ 87

$K_{\mathbf{G}}$ 20

$K_{\mathbf{g}_{\hat{i}}}$ 88

$K_{\mathbf{H}}$ 13

$K_{\widetilde{\mathbf{H}}}$ 19, 20

$K_{\mathbf{\Phi}}$ 13

L the connection function of a CSN 6, 11, 12, 14, 15, 17, 22, 23, 25, 54–56, 96

L the connection matrix of a CSN 12, 13, 18, 83

lcm least common multiple 46, 54, 56–58

M a CODESUB 11, 24–27, 86, 90, 91

M_i the CODESUB labeled i . . . . .

M_1 13–16

M_2 13–16

M_3 . . . . . 15, 16

M the function labeling CODESUBs 11, 14, 15

$\mathcal{M}$ the set of all possible CODESUBs 11, 24, 25, 27, 90, 91

N_{A} the number of SDFAs in the SDFG 31, 46, 47, 54–58, 67–73

N^{B} the number of SDFBs in the SDFG 31, 46, 55

networkToDataflow the function that constructs a CSW . 53, 56, 58, 59, 78, 80, 84, 86, 87, 90, 94

networkToEstimation the function that constructs a DEW 90, 96–98

N_{F} the number of CSLs in the CSN 53–56, 67–75, 77, 90

N_{M} the number of CODESUBs in the CODESYS 12

$\mathbb{N}$ the set of natural numbers . 11, 12, 21, 22, 30–36, 39, 40, 43–46, 50, 52–54, 57, 58, 62, 63, 65, 67, 76, 87, 89, 93, 94, 97

$\mathbb{N}_0$ the set of non-negative integer numbers 21, 31, 35, 41, 46, 53, 62, 78, 79, 93–95, 97

N_{U} the number of inputs of all CODESUBs 12, 13, 18

$N_{\mathbf{u}}$ the number of inputs of a CODESUB 11, 24

$N_{\mathbf{u}_i}$ . . . of the i^{th} CODESUB 96

N_{U} the number of input ports of a CSL 21, 25, 51, 89, 90

N_{U_i} . . . of the CSL labeled i 53–55

N_{U_j} 54, 55

num the number of operations that are executed in a particular SDFG execution step . . 35–39, 41–43, 63, 64

num$'$ 39–41

num$''$ 39–41

$N_{\mathscr{U}}$ the number of input ports of an SDFA 30, 31, 51, 89

$N_{\mathscr{U}_a}$ . . . of the SDFA labeled a 31, 54, 55, 57, 74, 75

$N_{\mathbf{X}}$ the number of states of all CODESUBs 12, 13, 18

$N_{\mathbf{x}}$ the number of states of a CODESUB 11

$N_{\mathbf{x}_i}$ . . . of the i^{th} CODESUB 18

$N_{\mathbf{Y}}$ the number of outputs of all CODESUBs 12, 18, 90

$N_{\mathbf{y}}$ the number of outputs of a CODESUB 11, 24

$N_{\mathbf{y}_i}$ . . . of the i^{th} CODESUB 18, 96

N_{Y} the number of output ports of a CSL 21, 51, 52

$N_{\mathscr{Y}}$ the number of output ports of an SDFA 30, 31, 51, 52, 89, 90

$N_{\mathscr{Y}_a}$ . . . of the SDFA labeled a 31, 54, 55, 57, 75

$\mathscr{O}$ big-O 26, 27, 87, 88, 91, 92

ω the function providing a value of a token in the SDFB 35–37, 41–43, 45, 47, 48, 50, 62, 63

ω_0 the function assigning initial tokens to input ports of SDFAs 31, 34, 36, 41, 43, 45, 47, 53, 54, 56, 58, 59, 63, 78, 80, 84–87, 90, 93, 94, 96–98

ω_{01} 65, 94

ω_{02} 66, 94

ω_{03} 94, 95

ω_0' 93

ω_{0opt}^H the optimal initial tokens obtained by minimization of the quality criterion on a single iteration 93

ω_{total} the sequence of tokens that visited an SDFB 41–43, 78

ω_{total}' 78, 79

operation the sequence of operations during the execution of an SDFG 35–43, 63, 64

operation$'$ 39, 40, 44

operation$''$ 39, 44

p the function that assigns a production rate to the output port in an SDFA . 30, 31, 51, 52, 89

p_a . . . in the SDFA labeled a 37, 44, 46, 57, 75

p_1 32

p_2 33

$p_{\hat{a}}$ 38, 39, 41–43, 63, 64, 76, 77

$p_{\hat{a}_0}$ 40

$p_{\underline{a}}$ 56, 75, 76, 85, 86

$p_{a'}$ 70, 71, 75

$p_{\hat{\imath}}$ 56

precedences the function enumerating the preconditions for the invocation of an SDFA 63, 65, 75–77

produce the function that carries out the production of tokens 35–43, 45, 47, 62–64

$\mathbb{Q}_{>0}$ the set of positive rational numbers 51, 53, 55–57, 66, 78, 87–89, 93, 96–98

$\mathbb{Q}$ the set of rational numbers 32

R the period of a PASS 45, 47–50

r_a the number of times the SDFA with label a is executed during an iteration 46, 47, 50, 57, 58, 67–69, 71–75, 77

r_1 46, 72

r_2 46

$r_{a''+1}$ 73

$r_{\breve{a}}$ 69–73

$r_{a'}$ 68–71

r_i 58

$r_{\breve{\imath}}$ 84–86, 88

$r_{N^{\mathrm{A}}}$ 46

$r'_{a'}$ 63

$\mathbf{r}$ the repetition vector of an SDFG 46–48, 57, 63, 67

$\mathbf{r}'$ the repetition vector of an SDFG 47

$\mathbb{R}_{\geqslant 0}$ the set of non-negative real numbers 11, 12, 18, 21, 23

$\mathbb{R}_{>0}$ the set of positive real numbers 21, 24, 25, 27, 62, 65, 84, 98

$\mathbb{R}$ the set of real numbers 11–13, 18, 24, 33, 93

S a CODESYS 11, 14, 15, 25–28, 84, 86, 87, 96

set the set function of an co-simulation slave 21, 22, 24

σ the PASS of an SDFGs 45, 47–49

slaveToEstimator the function used to obtain an ODEST 89–91

slaveToSimulator the function used to obtain a simulator from a CSL 51, 54, 86, 89

solver a function solving a CODESUB 24, 25, 86, 90, 91

src the source port providing the tokens to the buffer 31, 34, 37–41, 43, 46, 53–56, 63–73, 76, 90

$\mathcal{S}$ the set of all possible CODESYSs 25, 87, 96

subsystemSolvers a function that assigns a solver to each CODESUB in a CODESYS. 25–28, 84, 86, 87, 96

subsystemToSlave a function that constructs a CSL from a CODESUB 24, 25, 86, 90, 91

systemToNetwork a function that constructs a CSN from a CODESYS . 25–28, 84, 86, 87, 96

T the completion time of an SDFG task executed on the IPEP 65, 74–77

t a sequence of communication points assigned to a CSL 21, 25–28, 91

t_i ...to the i^{th} CSL 58

t_{end} the duration of a co-simulation experiment 78–81, 93, 96–98

time the execution time of the calculate function call 62, 65, 66, 74, 75

$T_{iteration}$ the completion time of an SDFI executed on the IPEP 65, 66, 77

t_κ the κ^{th} point in simulated time when discontinuity in the numerical solution can occur . 19, 20

t_0 the initial point in simulated time when discontinuity in the numerical solution can occur 19, 20

$t_{\kappa-1}$ the $\kappa - 1^{st}$ point in simulated time when discontinuity in the numerical solution can occur 19, 20

tol the defect tolerance specified for the automatic configuration algorithm........ 94, 97, 98

$\mathbf{U}$ the aggregation of input signals of all CODESUBs 12, 13, 18, 19

$\mathbf{u}$ the input signal of a CODESUB 11, 12

$\mathbf{u}_i$ ...of the i^{th} CODESUB 6, 11

$\mathbf{u}_1$ 12, 14, 16

$\mathbf{u}_2$ 12, 14, 16

$\mathbf{u}_3$ 16

$\mathbf{u}_N$ 12

$\underline{\mathbf{u}}$ the input signal of a SUBW 26–28

$u_{\breve{\imath}\breve{\imath}}$ the signal consumed by the $\breve{\imath}^{th}$ port of the $\breve{\imath}^{th}$ CODESUB 6, 12

$\widetilde{\mathbf{U}}$ the numerical solution found for the input signal of all CODESUBs............ 18, 19, 83

$\widetilde{\mathbf{u}}$ the numerical solution found for the input signal of a CODESUB 25, 86, 87

$\widetilde{\mathbf{u}}'$ the numerical solution found for the input signal by an ODEST 92

$\widetilde{u}_{\breve{\imath}\breve{\imath}}$ the numerical solution for the signal consumed by the $\breve{\imath}^{th}$ port of the $\breve{\imath}^{th}$ CODESUB 6, 17, 18, 84, 85

$\widetilde{\mathbf{u}}_i$ the numerical solution found for the input signal of the i^{th} CODESUB 17, 87, 88, 91

$\widetilde{\mathbf{u}}_1$ 83

$\widetilde{\mathbf{u}}_2$ 83

u the sequence of input values assigned to the ports of a CSL.................. 21, 24, 51

u' ...wrapped with an ODEST... 89, 90

$\underline{u}_\iota$ the input signal of a SUBW at the port labeled ι 21

$\underline{u}_{\breve{\imath}}$ 23

u_ι the input signal at the input port indexed i 12

- u_1 12
- u_2 12
- $u_{N_\mathbf{u}}$ 12

U the function labeling input port sets of a CSL 21, 22, 24, 51

$\mathscr{U}$ the function labeling input port sets of an SDFA 30, 51, 52, 89

- $\mathscr{U}_a$ . . . of the SDFA labeled a . . . . .
  - $\mathscr{U}_1$ 32
  - $\mathscr{U}_2$ 33

$\mathcal{U}$ the set of sets of input values 21, 30, 62

v_0 the initial state of a CSL 21, 22, 24

- v_{0i} . . . of the co-simulation slave labeled i 54
  - v_{01} 59, 80
  - v_{02} 59, 80

V the set of internal states of an CSL 21, 22, 24, 25, 27, 51

v the sequence of states of a CSL 21, 25, 27, 28, 51, 52

- v' . . . ODEST 89, 90

$\mathbf{x}_0$ the initial state of a CODESUB 11, 24

- $\mathbf{x}_{0i}$ . . . the i^{th} CODESUB 6, 11, 17, 26
  - $\mathbf{x}_{01}$ 14, 16
  - $\mathbf{x}_{02}$ 14, 16
  - $\mathbf{x}_{03}$ 16

$\mathbf{X}$ the aggregation of state signals of all CODESUBs 12, 13, 18, 19

$\mathbf{x}$ the state signal of a CODESUB 11, 13

- $\mathbf{x}_i$ . . . of the i^{th} CODESUB 6, 11
  - $\mathbf{x}_1$ 12, 14, 16
  - $\mathbf{x}_2$ 12, 14, 16
  - $\mathbf{x}_3$ 16
  - $\mathbf{x}_N$ 12

$\mathbf{X}_0$ the initial state of all CODESUBs 12

$\dot{\mathbf{x}}$ the derivative of a CODESUB state signal 11

- $\dot{\mathbf{x}}_i$ . . . of the i^{th} CODESUB 6, 11
  - $\dot{\mathbf{x}}_1$ 14, 16
  - $\dot{\mathbf{x}}_2$ 14, 16
  - $\dot{\mathbf{x}}_3$ 16

$\underline{\dot{\mathbf{x}}}$ the derivative of the state signal of a SUBW . . . . .

- $\underline{\dot{\mathbf{x}}}_i$ . . . of the SUBW labeled i 26

$\underline{\mathbf{x}}$ the state signal of a SUBW 26–28

  $\underline{\mathbf{x}}_i$ . . . of the SUBW labeled i 26

$\dot{\mathbf{X}}$ the aggregation of signal derivatives of all CODESUBs states 12, 13

$\dot{\tilde{\mathbf{x}}}_i$ the derivative of the solution for the i^{th} CODESUB state signal 6, 17

  $\dot{\tilde{\mathbf{x}}}_1$ 83

  $\dot{\tilde{\mathbf{x}}}_2$ 83

$\widetilde{\mathbf{X}}$ the numerical solution found for the state signal of all CODESUBs 18–20

$\tilde{\mathbf{x}}$ the numerical solution found for the state signal a CODESUB 25, 87

  $\tilde{\mathbf{x}}_i$ . . . of the i^{th} CODESUB 6, 17, 18, 87, 88, 91

    $\tilde{\mathbf{x}}_1$ 83

    $\tilde{\mathbf{x}}_2$ 83

    $\tilde{\mathbf{x}}_{\hat{\imath}}$ 88

$\tilde{\mathbf{x}}'$ the numerical solution found for the state signal by an ODEST 91, 92

ξ the function referencing each individual input token consumed by an SDFA 30, 51, 52

  ξ_a . . . the SDFA labeled a 36, 44, 45, 47, 62

    ξ_1 33

    ξ_2 33

    $\xi_{\breve{a}}$ 43

    $\xi_{\underline{a}}$ 85

    ξ'_a 44

    ξ''_a 44

    $\xi_{\breve{\imath}}$ 85

  ξ' . . . ODEST 89, 90

$\mathbf{Y}$ the aggregation of output signals of all CODESUBs 12, 18

$\mathbf{y}$ the output signal of a CODESUB 11

  $\mathbf{y}_i$ . . . the i^{th} CODESUB 6, 11

    $\mathbf{y}_1$ 12, 14, 16

    $\mathbf{y}_2$ 12, 14, 16

    $\mathbf{y}_3$ 16

    $\mathbf{y}_N$ 12

$\underline{\mathbf{y}}$ the output signal of a SUBW . . . . . .

  $\underline{\mathbf{u}}_i$ . . . of the SUBW labeled i 26

  $\underline{\mathbf{y}}_i$ . . . of the SUBW labeled i 26

$\widetilde{\mathbf{Y}}$ the numerical solution found for the output signal of all CODESUBs 18, 83

$\tilde{\mathbf{y}}$ the numerical solution found for the output signal of a CODESUB . . . . . .

  $\tilde{\mathbf{y}}_i$ the numerical solution found for the output signal of the i^{th} CODESUB 6, 17, 18

    $\tilde{\mathbf{y}}_1$ 83

84, 89, 94, 97

Biography

Slaven Glumac was born on the 22nd of March 1989 in Bjelovar, Croatia. In 2007, he enrolled the Faculty of Electrical Engineering and Computing, where he completed the undergraduate study in electrical engineering and information technology. On the same faculty in 2012 he graduated in the field of electrical engineering and information technology with the topic “Intelligent control of elephant trunklike robot”, under the mentorship of Professor Zdenko Kovačić.

From February 2013 to December 2014 he worked as a teaching assistant in the mechatronics department of the Technical College in Bjelovar. There he was actively involved in several undergraduate courses in the areas of electrical engineering, signal processing, control theory and software programming. From January 2015 to June 2021 he worked as a software development engineer at AVL-AST d.o.o. There he developed software for test automation systems in the automotive industry. The main focus of his work was the integration of simulations into the automation systems. Since June 2021 he has been working as a software architect at Spyrosoft Solutions d.o.o. His research interests include real-time simulation, co-simulation, optimization methods, model-based development of physical and software systems. He was involved in the publication of five conference papers, one journal paper and a chapter in a book on robotic manipulators.

List of published works

Papers in journals

1. Glumac, S., Varga, N., Raos, F., Kova čić, Z., “Co-simulation perspective on evaluating the simulation with the engine test bench in the loop”, Automatika, Vol. 63, Issue 2, January 2022, pp. 275-287.

Papers at international scientific conferences

1. Glumac, S., Kova čić, Z., “Relative consistency and robust stability measures for sequential co-simulation”, Proceedings of the 13th International Modelica Conference, March 2019, pp. 197-206.
2. Glumac, S., Kova čić, Z., “Calling sequence calculation for sequential co-simulation master”, Proceedings of the 2018 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation, May 2018, pp. 157-160.
3. Glumac, S., Arbula, D., Kova čić, Z., “Microimmune algorithm for sensor network localization”, 2015 IEEE Sensors Applications Symposium, April 2015, pp. 1-6.
4. Ko čo, E., Glumac, S., Kovačić, Z., “Multiobjective optimization of a quadruped robot gait”, 22nd Mediterranean Conference on Control and Automation, June 2014, pp. 1520-1526.
5. Glumac, S., Kova čić, Z., “Microimmune algorithm for solving inverse kinematics of redundant robots”, 2013 IEEE International Conference on Systems, Man, and Cybernetics, October 2013, pp. 201-207.

Životopis

Slaven Glumac rođen je 22. ožujka 1989. u Bjelovaru, Hrvatska. 2007. godine upisao je Fakultet elektrotehnike i računarstva, gdje je završio preddiplomski studij elektrotehnike i informacijske tehnologije. Na istom fakultetu 2012. diplomirao je na području elektrotehnike i informacijske tehnologije s temom "Inteligentno upravljanje dvoručnim polusurlastim robotom", pod mentorstvom profesora Zdenka Kovačića.

Od veljače 2013. do prosinca 2014. radio je kao asistent u nastavi na odsjeku mehatronike Visoke tehničke škole u Bjelovaru. Tamo je bio aktivno uključen u nekoliko preddiplomskih kolegija iz područja elektrotehnike, obrade signala, teorije upravljanja i programiranja. Od siječnja 2015. do lipnja 2021. radio je kao inženjer za razvoj programske podrške u tvrtki AVL-AST d.o.o. Tamo je razvijao programsku podršku za sustave automatizacije ispitivanja u automobilskoj industriji. Glavni fokus njegovog rada bila je integracija simulacija u sustave automatizacije. Od lipnja 2021. zaposlen je kao arhitekt programske podrške u tvrtki Spyrosoft Solutions d.o.o. Njegovi istraživački interesi uključuju simulaciju u stvarnom vremenu, kosimulaciju, metode optimizacije, razvoj fizikalnih i programskih sustava temeljen na modelima. Sudjelovao je u objavljivanju pet konferencijskih radova, jednom radu u časopisu i jednom poglavlju u knjizi o robotskim manipulatorima.