SYNTHESIZING CO-SIMULATION ALGORITHMS WITH STEP NEGOTIATION AND ALGEBRAIC LOOP HANDLING

Simon Thrane Hansen
Cláudio Gomes
Peter Gorm Larsen

DIGIT, Department of
Electrical and Computer Engineering,
Aarhus University
Finlandsgade 22, Aarhus, Denmark
{sth, claudio.gomes, pgl}@ece.au.dk

Jaco van de Pol

DIGIT, Department of Computer Science
Aarhus University
Åbogade 34, Aarhus, Denmark
jaco@cs.au.dk

ABSTRACT

Simulation-based analyses of cyber-physical systems are becoming increasingly vital. Co-simulation is one such technique that enables the coupling of specialized simulation tools through an orchestration algorithm that dictates when, with which inputs, and how far each simulation tool should advance its corresponding subsystem. The result of a co-simulation is often sensitive to the orchestration algorithm. This algorithm should be designed based on the simulation tool's implementation to minimize the co-simulation error. The paper extends current graph-based approaches for generating implementation-aware simulation algorithms to cover complex co-simulation scenarios where step negotiation and algebraic loops are part of the simulation. We show how the generation takes place for different complex co-simulation scenarios. An implementation of the approach is available online.

Keywords: Co-Simulation, Orchestration Algorithms, Synthesis of Algorithms.

1 INTRODUCTION

Co-simulation is a simulation technique that allows simulating a complete system consisting of multiple black-box simulation units (SUs), each responsible for computing the behavior of a sub-system, cf. (Kübler and Schiehlen 2000, Gomes et al. 2018). Co-simulation is often used in developing cyber-physical systems (CPS), where the different sub-systems typically are developed separately using different tools and techniques. Co-simulation allows iterative integration of simulation units to let all interested parties explore the global system behavior through the entire development cycle. The SUs are coupled using an orchestrator algorithm that exchanges values between SUs, using a standardized interface. The interface protects the SU's intellectual property and provides functionality for setting/getting inputs/outputs and computing the associated model behavior over a given interval of time. The Functional Mockup Interface (FMI) Standard ((Blockwitz et al. 2012), FMI 2014) is one example of such an interface. FMI inspires our notion of an SU.

Researchers have shown empirically in Gomes et al. (2019) that co-simulation results can be highly sensitive to the orchestrator algorithm used. Earlier, the works in (Busch 2016, Kalmar-Nagy and Stanciulescu 2014, Schweizer, Li, and Lu 2015, Arnold 2010, Gomes et al. 2018) introduced techniques to study the stability

of different co-simulation algorithms. The presented empirical studies reinforce the fact that choosing the appropriate orchestrator algorithm is important. In particular, Gomes et al. (2018) shows how orchestrator algorithms that correctly handle algebraic loops results in a stable co-simulation, for a wide r range of system parameters.

To unlock the potential of co-simulation, the correctness of the result needs to be guaranteed. The above studies highlight the importance of a co-simulation algorithm that is aware of some implementation details of each SU to obtain such a result. For instance, ignorance of an SU's input approximation function may lead to "hard to debug errors" in the co-simulation results. Constraints on the co-simulation algorithm can be imposed based on information provided by the user, leading to improved co-simulation results (Gomes et al. 2019, Oakes et al. 2020). These constraints have been formally encoded in a tool (https://github.com/INTO-CPS-Association/Scenario-Verifier), making it possible to verify a co-simulation algorithm. However, currently, no tools provide the ability to generate implementation-aware algorithms for complex co-simulation scenarios that enables adaptive step simulation for stiff systems and fixed-point iteration to handle algebraic loops. The tailored algorithms are more suitable for verification and have increased performance compared to a generic algorithm.

Contribution. This paper proposes an approach to derive implementation aware co-simulation algorithms for scenarios subject to step negotiation and algebraic loop iteration, which an existing tool can automatically verify. The technique requires that all SUs allow their state to be restored, which, at the time of writing, is not supported by many FMUs. The approach extends the work of Gomes et al. (2019) and has been tested on numerous case studies, including an industrial case study from Boeing presented in Gomes, Lucio, and Vangheluwe (2019). The focus is on co-simulation scenarios that include many continuous time SUs, since these are the ones that usually benefit from better input approximation techniques. Discrete time SUs that represent, e.g., software components, usually default to sample-and-hold extrapolations, which, as will later become clear, result in trivial co-simulation scenarios.

Organization. The next section introduces the concept of co-simulation and co-simulation algorithm, along with a presentation of previous techniques for synthesizing co-simulation algorithms. Sect. 3 describes our contribution. Sect. 4 describes the related word. Finally, Sect. 5 concludes and provides an outlook of future work.

2 BACKGROUND

This section presents the concept of co-simulation and a formalization of SUs and an introduction to state of the art techniques for synthesizing co-simulation algorithms. The formalization is adapted from Broman et al. (2013), Gomes et al. (2019).

Co-simulation is a technique enabling the global simulation of a system consisting of multiple black-box SUs typically developed individually or exported from different tools. An SU captures the behavior of a dynamical system, a function from time and space into some often multi-dimensional and continuous space. The dynamical system engages with the environment through inputs and outputs (Kübler and Schiehlen 2000,Gomes et al. 2018). SUs are coupled through their inputs and outputs, indicating that the state of one SU relies on the state of another SU at all times - known as a coupling restriction. An SU has its own solver/simulator that can calculate the model's behavior trace at any given time based on the couplings to other SUs. The coupling restrictions can, in practice, only be satisfied at specific points in time referred to as communication points. Between the communication points, where each SU updates its behavior trace, an SU makes assumptions about the evolution of the value on its inputs. These assumptions can cause a significant error in the co-simulation for a considerable interval and are the main cause of errors (Arnold, Clauß, and Schierz 2014).

Definition 1. *An SU with identifier c is represented by the tuple* $\langle S_c, U_c, Y_c, \texttt{set}_c, \texttt{get}_c, \texttt{doStep}_c \rangle$, *where:* S_c *represents the state space.* U_c *and* Y_c *the set of input and output variables, respectively.* $\texttt{set}_c : S_c \times U_c \times \mathcal{V} \to S_c$ *and* $\texttt{get}_c : S_c \times Y_c \to \mathcal{V}$ *are functions to set the inputs and get the outputs, respectively (we abstract the set of values that each input/output variable can take as* $\mathcal{V}$*).* $\texttt{doStep}_c : S_c \times \mathbb{R}_{>0} \to S_c \times \mathbb{R}_{>0}$ *is a function that instructs the SU to compute its state after a given time duration. If an SU is in state* $s_c^{(t)}$ *at time t,* $(s_c^{(t+h)}, h) = \texttt{doStep}_c(s_c^{(t)}, H)$ *approximates the state* $s_c^{(t+h)}$ *of the corresponding model at time* $t+h$*, with* $h \leq H$.

The function $\texttt{doStep}_c$ returns a step size because some SUs implement error estimation and may conclude that taking a step size of H will result in an intolerable error. In Sect. 3, we address the consequences of such behavior.

Definition 2 (Scenario)**.** *A scenario is a structure* $\langle C, L, M, D, R \rangle$ *where each identifier* $c \in C$ *is associated with an SU, as defined in Definition 1, and* $L(u) = y$ *means that the output y is connected to input u. Let* $U = \bigcup_{c \in C} U_c$ *and* $Y = \bigcup_{c \in C} Y_c$*, then* $L : U \to Y$*. If an SU c may reject to perform a step of an arbitrary duration this is denoted by* $c \in M$*. Naturally* $M \subseteq C$*. The orchestrator and SUs agree on the communication points for the scenario; the orchestrator's goal is to find the communication points that minimize the error introduced in the co-simulation.* $R = \bigcup_{c \in C} R_c$*, where* $R_c(u_c) = true$ *means the function* $\texttt{doStep}_c$ *assumes that the input* u_c *comes from an SU that has advanced forward relative to SU c. And* $D = \bigcup_{c \in C} D_c$*, where the input* $u_c \in U_c$ *feeds through to output* $y_c \in Y_c$*, that is,* $(u_c, y_c) \in D_c$*, when there exists* $v_1, v_2 \in \mathcal{V}$ *and* $s_c \in S_c$*, such that* $\texttt{get}_c(\texttt{set}_c(s_c, u_c, v_1), y_c) \neq \texttt{get}_c(\texttt{set}_c(s_c, u_c, v_2), y_c)$.

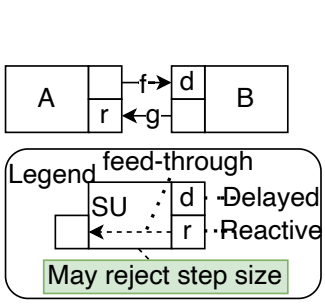
This paper uses the syntax in Fig. 1a to graphically present co-simulation scenarios. Definition 2 is based on studies (Gomes et al. 2019, Oakes et al. 2020, Gomes et al. 2018, Schweizer et al. 2015, Gomes et al. 2019) that conclude that the optimal communication points depend on the implementation of the SUs. These implementation details of the SUs are encoded in Definition 2 as the reactivity, feed-through, and step rejection constraints. We define an implementation-aware algorithm as an algorithm respecting all the constraints dictated by the scenario.

The simulation of a scenario is controlled by an orchestrator, an algorithm trying to satisfy the coupling restrictions and ensure that the SUs progress in time in lockstep. As we will explain below, the feed-through D, reactivity R, and step size rejection M, are used to constrain the candidate orchestration algorithms.

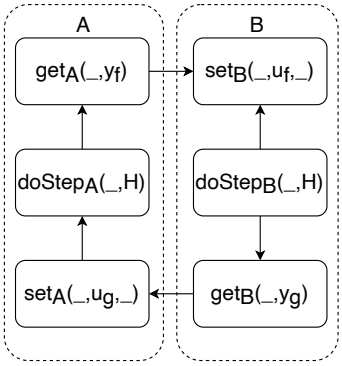
An orchestration algorithm consists of both an initialization procedure and a co-simulation step procedure. The initialization procedure initializes the system, while the step procedure runs the simulation. An example of a step procedure is Algorithm 1 in the appendix of a simple scenario where none of the SUs can reject a step of size H. We refer to SUs in algorithms using capital letters (i.e. "A" and "B" in Algorithm 1), values are denoted by the subscript "val" (i.e. g_{val}) and ports are denoted using the subscript of their label. All algorithms throughout the paper are in the appendix. This paper focuses on the step procedure since an initialization procedure is a simpler version of the former.

Definition 3 (Step)**.** *Given a scenario* $\langle C, L, M, D, R \rangle$*, a co-simulation step is a finite ordered sequence of SU function calls* $(f_i)_{i \in \mathbb{N}} = f_0, f_1, \ldots$ *with* $f_i \in F = \bigcup_{c \in C} \{\texttt{set}_c, \texttt{get}_c, \texttt{doStep}_c\}$*, and i denoting the order of the function calls.*

Gomes et al. (2019) has developed a graph-based approach for synthesizing co-simulation algorithms for simple co-simulation scenarios. It relies on the fact that each SU operation should only be executed once for simple co-simulation scenarios (Gomes et al. 2019, Corollary 1). The approach builds a graph of the operations of the co-simulation step. The graph's topological order denotes the order of the function calls.



(a) Example scenario.



(b) The step operation graph.

Figure 1: Example co-simulation scenario, its step operation graph, and the Step procedure.

The order of the function calls of a co-simulation step can be derived based on the constraints of the SU operations described in Gomes et al. (2019).

Definition 4 (Step Operation Graph)**.** *Given a co-simulation scenario $\langle C, L, M, D, R\rangle$, we define the step operation graph where each node represents an operation* $\texttt{set}_c(\_, u_c, \_)$*,* $\texttt{doStep}_c(\_, H)$*, or* $\texttt{get}_c(\_, y_c)$*, of some fmu $c \in C$, $y_c \in Y_c$, and $u_c \in U_c$. The edges are created according to the following rules:*

1. *For each $c \in C$ and $u_c \in U_c$, if $L(u_c) = y_d$, add an edge* $\texttt{get}_d(\_, y_d) \rightarrow \texttt{set}_c(\_, u_c, \_)$*;*
2. *For each $c \in C$ and $y_c \in Y_c$, add an edge* $\texttt{doStep}_c(\_, H) \rightarrow \texttt{get}_c(\_, y_c)$*;*
3. *For each $c \in C$ and $u_c \in U_c$, if $R_c(u_c) = true$, add an edge* $\texttt{set}_c(\_, u_c, \_) \rightarrow \texttt{doStep}_c(\_, H)$*;*
4. *For each $c \in C$ and $u_c \in U_c$, if $R_c(u_c) = false$, add an edge* $\texttt{doStep}_c(\_, H) \rightarrow \texttt{set}_c(\_, u_c, \_)$*;*
5. *For each $c \in C$ and $(u_c, y_c) \in D_c$, add an edge* $\texttt{set}_c(\_, u_c, \_) \rightarrow \texttt{get}_c(\_, y_c)$*.*

This approach by Gomes et al. (2019) has been used with success for simple scenarios, like the one shown in Fig. 1a. Note that item 4 in Definition 4 does not represent a data dependency, but it is added to respect the contract on the delayed input. Algorithm 1 is derived from the graph in Fig. 1b. However, the method cannot derive algorithms for complex scenarios, with algebraic loops (see Fig. 2a) or where step negotiation (see Fig. 4a) is needed.

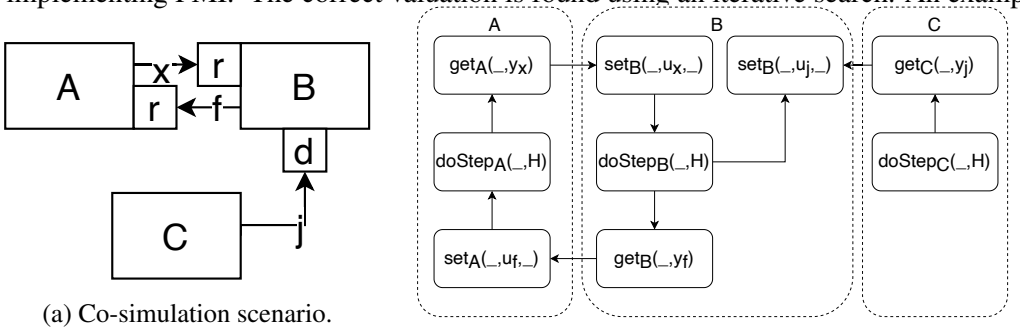
3 CO-SIMULATION OF COMPLEX SCENARIOS

This section presents the approach for simulating complex co-simulation scenarios. A brief background is given on the definition and simulation of complex scenarios. Afterwards, our approach to generate orchestrators is presented using different examples of complex scenarios.

3.1 Complex co-simulation scenario

A complex co-simulation scenario is a co-simulation scenario containing either an algebraic loop (i.e. a cyclic dependency) or where at least one SU c in the scenario may reject to perform a step of arbitrary size (meaning $c \in M$). Fig. 2a shows a complex scenario with an algebraic loop. Complex scenarios are challenging to simulate because the orchestration algorithm needs to satisfy the constraints associated with each SU, account for possible rejections of steps, solve algebraic loops. As we detail below, this is achieved by finding a correct valuation of all inputs and outputs in the scenario, that ensures that all SUs agree on the step duration and that satisfies all algebraic loops. This valuation needs to be found in every co-simulation step. All the examples in this paper assume the existence of a correct valuation. We also assume that all SUs of a complex scenario allow their state to be restored, as this is a requirement to solve algebraic loops

with reactive ports and perform step negotiation. At the time of writing, this is not a feature supported by many SUs implementing FMI. The correct valuation is found using an iterative search. An example of this



(a) Co-simulation scenario.

(b) Step operation graph.

Figure 2: Example of complex co-simulation scenario.

approach is shown by Algorithm 2 where a fixed-point is established on ports x and f. The iterative search potentially requires restoring the state of each of the SUs involved in the search to restart the search. The general way to add state restoration is to save all SUs before the iterative search and restore their state if a correct valuation has not been found, before trying again. Since these state save and restore operations always follow the same pattern, they are omitted from the step operation graph.

3.2 Handling of Algebraic loops

An algebraic loop indicates that the value retrieved on an output port depends on itself as shown in Fig. 2b. In the figure, SU A can only step to time $t+H$ and compute x after it receives a value of f at time $t+H$. SU B has an analogous restriction, so the two SUs are deadlocked. Algebraic loops can be identified by building a step operation graph according to Definition 4, and applying Tarjan's algorithm (Tarjan 1972). Any non-trivial strongly connected component (SCC) in the graph highlights an algebraic loop. There exist two kinds of algebraic loops, as identified in Kübler and Schiehlen (2000), Fig. 5 and 6): Feed-through and Reactivity loops. They differ in nature, but the way to simulate them and their impact on the scenario are very similar. We start by describing the technique for solving reactivity loops since feed-through loops can be seen as a simplified case of reactive loops.

Reactivity loops occur when multiple SUs rely upon other SUs to be advanced in time relative to itself. They occur in a scenario due to reactivity constraints between SUs like the scenario in Fig. 2a. A reactivity loop is identified as a non-trivial SCC in the graph formed by Definition 4 that adheres to Definition 5.

Definition 5 (Reactivity SCC)**.** *A non-trivial SCC of the step operation graph of Definition 4 is a reactivity SCC when it contains at least one node* `doStep_`$(\_,\_)$.

Fig. 2a shows a scenario with a reactivity loop. The cyclic step operation graph of the example is shown in Fig. 2b. The algorithm to simulate the scenario in Algorithm 2 contains a fixed-point iteration procedure. The technique for synthesizing these algorithms is described next.

3.2.1 Synthesizing Algebraic Loop Solvers

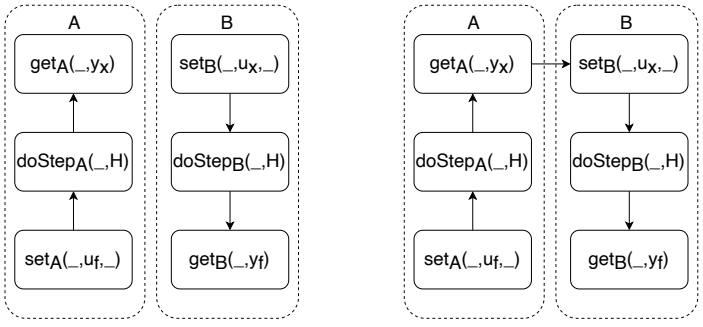
This section presents a method for generating a co-simulation algorithm to find a fixed-point of an algebraic loop. We show the approach using the scenario in Fig. 2a as an example. The correct valuation of such a scenario is to reach a fixed-point on all outputs that match the corresponding inputs. We check for convergence by comparing the output values between two successive iterations, as done in Algorithm 2.

To generate the operations that comprise each iteration, we need to make the graph acyclic, so an ordering of SU operations can be derived. The graph is made acyclic using reductions. The reductions work by looking at a relaxed version of the scenario where some couplings between SUs are removed. The relaxation is tolerated because the generated orchestration algorithm will provide an informed guess on the input values whose coupling is not respected. The guesses are supplied in the first iteration on lines 4 and 5 in Algorithm 2. This relaxation allows removing some of the edges from the step operation graph. There are two kinds of reductions, minimum and maximum. The user should preferably supply the type of reduction used to synthesize the algorithm. It is outside the scope of this paper to determine which reduction scheme works best, as this is highly dependent on the co-simulation scenario.

It is important to note that the reductions will cause the generated algorithm to violate the SU contracts as long as the iteration has not converged. The key insight of our work is the fact that on the last iteration, when the convergence check succeeds, all SU contracts will be satisfied, because the values set on the input ports are now close enough to those obtained from the current output port at the current point in time. For example, this is enabled by the assignments in line 15 in Algorithm 2 that update the variables x and f to a progressed value compared to the SUs A and B, which have been restored.

Maximum Reduction. The maximum reduction technique is inspired by the Jacobi method for solving a system of linear equations. The reduction works by providing a guess for the value set on all inputs in the loop. The edges we remove depends on the type of the algebraic loop. We only remove edges to reactive input ports for reactivity loops, while, for feed-through loops, we remove all edges to input ports, including delayed input ports. By applying the maximum reduction technique to the graph of Fig. 2b we obtain the graph in Fig. 3a. This graph is acyclic, and from it the co-simulation algorithm in Algorithm 2 is calculated.

Minimum Reduction. The minimum reduction technique is inspired by the Gauss-Seidel method for solving a system of linear equations. The approach works by providing guesses for the values to the minimum number of input ports to break the loop. Ideally, the user (with insight in the internal working of the SUs) should decide how to break the loop since it could severely impact how quickly a fixed-point is obtained. If the user does not provide any hints, the chosen approach is to provide guesses for the minimum number of inputs to make the graph acyclic. If the minimum reduction technique is applied to the graph in Fig. 2b, the graph in Fig. 3b is obtained.



(a) The Maximum reduced graph of Fig. 2b.

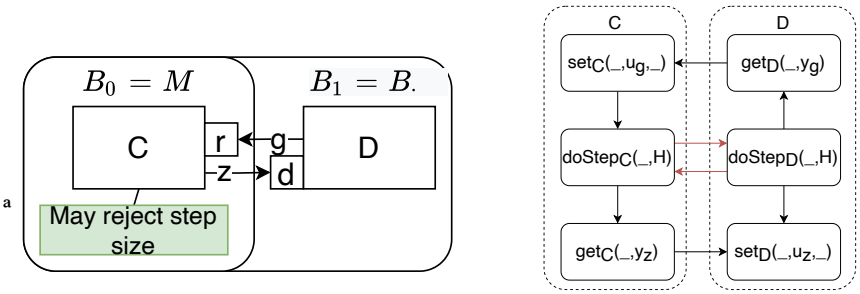
(b) The Minimum reduced graph of Fig. 2b.

Figure 3: Reduction schemes.

We conclude the discussion on algebraic loops by addressing feed-through loops. They are highlighted by non-trivial SCCs that are not reactivity SCCs according to Definition 5. Feed-through loops are also simulated using a fixed-point iteration procedure, which is obtained using reductions. However, since they do not involve any time progression, there is no need to save and restore the state of the SUs in the loop.

3.3 Step Negotiation

Step negotiation is needed if at least one of the SUs in the scenario may reject a step. In this case, the orchestrator algorithm should ensure that all SUs agree on a step size by performing step negotiation. Step negotiation can, in some cases, be avoided by using an appropriate fixed step. However, this step can be hard to determine for black-box SUs and complex physical systems or be inefficient for the simulation of stiff systems. Step negotiation is an iterative search for a step size that all SUs can perform. The negotiation procedure consists of a sequence of SU operations where the step size is shrunk to the smallest performed step of the previous iteration between unsuccessful iterations, until all SUs agree on a step. The procedure may require multiple attempts, meaning that restoration of state is needed. The method is inspired by Broman et al. (2013). Fig. 4a shows a scenario that needs step negotiation.



(a) Co-simulation scenario.

(b) The extended step operation graph. The red edges denote the new edges added by Definition 6.

Figure 4: A scenario needing step negotiation.

The problem of the original step operation graph (Definition 4) is that it does not reveal that step negotiation is needed. Therefore, we extend the original step operation graph with the rules of Definition 6.

The key reason behind introducing extra dependencies in the operation graph is that it allows deriving the step negotiation algorithm in a similar way as the algebraic loops, while at the same time respecting the reactivity and feed-through contracts of each SU, at the last iteration. With these changes, all SUs involved in the step negotiation will form a non-trivial SCC that will contain all the actions needed in the step negotiation procedure.

We define the set B of all the SUs that should be backtracked if an SU rejects a step. The set is iteratively calculated where the initial value of B is M. The set B is in each iteration updated with all SUs c that have a coupling to a reactive input of an SU d where $d \in B$. This ensures that all SUs that need to perform a `doStep`-operation before an SU that may reject a step are in B. B is iteratively expanded until B remains constant. Since the number of SUs in a scenario is finite, the update procedure is guaranteed to terminate. The calculation of the set B is shown in Fig. 4a.

Definition 6 (Step negotation extension of Definition 4)**.** *The step operation graph is extended by the following rules:*

6. *For each* $c1 \in C$ *and* $c2 \in C$ *where* $c1 \neq c2$ *and* $c1 \in B$ *and* $c2 \in B$ *add an edge from* $\texttt{doStep}_{c1}(\_,H) \rightarrow \texttt{doStep}_{c2}(\_,H)$ *;*
7. *For each* $c1 \in C$ *and* $c2 \in C$ *where* $c1 \in B$ *and* $c2 \notin B$ *add an edge from* $\texttt{doStep}_{c1}(\_,H) \rightarrow \texttt{doStep}_{c2}(\_,H)$.

Rule 6 ensures that all SUs that may reject to perform a step or are directly affected by another SU rejecting a step are part of a non-trivial SCC. This makes it possible to derive a step negotiation procedure and identify which SUs should be backtracked if one SU rejects a step. Item 7 reflects the fact that `c1` can accept any

step, but `c2` cannot. Therefore we need to know what step size can `c2` take before stepping `c1` to avoid having to restore the state of `c1`. It ensures that the step negotiation procedure is executed first in the step algorithm, so as few SUs as possible need to be restored if an SU rejects a step.

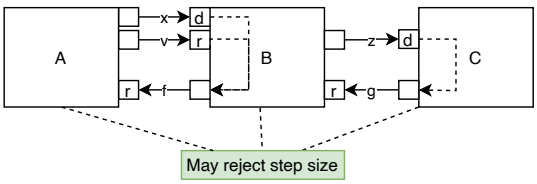
If the extra edges from Definition 6 are applied to the step operation graph. The graph of scenario Fig. 4a evolves to the graph presented in Fig. 4b which is not acyclic.

Definition 7 (Step-finding SCC)**.** *A step-finding SCC is an SCC in the extended step operation graph with an edge from one* `doStep_(_,_)` *node to another* `doStep_(_,_)` *node.*

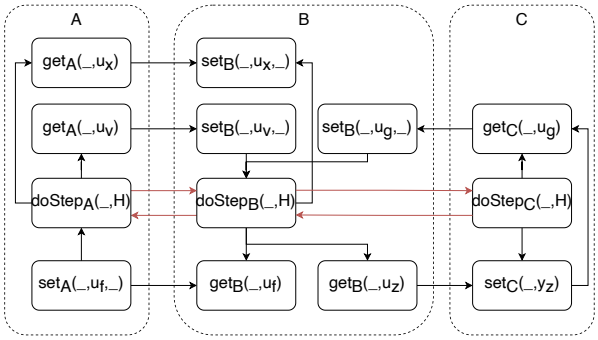
The extension of the graph can introduce an artificial cycle in the graph to show that the scenario needs step negotiation. After extending the graph by the rules of Definition 6, a step negotiation procedure can be identified as a non-trivial SCC that adheres to Definition 7. The step negotiation procedure is derived from the non-trivial SCC by removing the edges introduced by Item 6 of Definition 3 which in most cases make the graph acyclic. The step negotiation procedure shown in Algorithm 4 of the scenario is now derived as the topological order of the reduced non-trivial step-finding SCC.

3.4 Nested Step and Algebraic Loops

In most cases, removing some of the artificial edges of Item 6 in Definition 6 makes the step operation graph acyclic. However, in some scenarios the graph still contains cycles, even after removing these edges. In such cases, there exist both algebraic loops and step negotiation loops. Fig. 5a shows an example of such scenario. Nested complex scenarios are simulated using a step negotiation procedure that contains a fixed-point iteration procedure like in Algorithm 3. The algorithm is derived by first deriving the step negotiation procedure, and then the fixed-point iteration procedure of the reactivity loop inside the step negotiation cycle. We denote the set of SUs in the reactive loop in the scenario as K.



(a) A nested complex scenario.



(b) Step operation graph of Fig. 5a. The edges of Definition 6 are red.

Figure 5: A nested complex scenario.

3.5 Summary of the approach

This section summarises the approach presented using examples in the previous section. The orchestrator algorithm is derived using a graph-based approach where the graph is formed by the union of the rules of Definitions 4 and 6.

The graph is analysed to identify the topological order of the SCCs using Tarjan's algorithm (Tarjan 1972). The topological order of the SCCs is the correct order of the function calls in the co-simulation step. The

characteristics of each of the non-trivial SCC are analysed and handled accordingly. The approach has been implemented and is available (https://github.com/INTO-CPS-Association/Scenario-Verifier) as a part of a tool-chain for generating and verifying co-simulation algorithms.

The initialization procedure can also be derived using the presented approach on a simpler graph without $\texttt{doStep}_c(\_, H)$-nodes. The initialization procedure does therefore not need to perform step negotiation or solve reactivity loops. Hansen et al. (2021) describes this topic in more detail.

4 RELATED WORK

There have been some efforts to synthesize co-simulation algorithms that exhibit good stability properties automatically. We can classify these works into two main trends: those that assume that SUs can be adjusted to the orchestrator algorithm; and those that respect SU contracts.

In the first category, we place the works in Benedikt and Holzinger (2016), Holzinger and Benedikt (2019b), Holzinger and Benedikt (2019a), Oakes et al. (2020), Gomes et al. (2019), where the main theme is to formulate an optimization problem, whose cost function reflects an approximation of the error made by a specific choice of co-simulation algorithm. In contrast to our work, algebraic loops are broken up and not handled, but the choice of which inputs to guess is considered in the optimization problem. In our work, algebraic loops are handled, resulting in slower algorithms, but respecting the SU constraints.

The works in the second category assume that SUs cannot be adapted, and therefore their constraints must be respected: (Galtier et al. 2015, Hansen et al. 2021, Gomes et al. 2019, Broman et al. 2013). Prior work in Gomes et al. (2019) uses a graph-based approach to generate a co-simulation algorithm, for simple co-simulation scenarios. In Galtier et al. (2015), Hansen et al. (2021), the authors use the graph-based approach to perform the SCC computation and derive the initialization algorithm that handles feed-through algebraic loops. It is also interesting to mention the approach in Broman et al. (2013) where an interpreter is described that correctly respects the SU step size negotiation requirements, but does not handle feed-through constraints. In contrast, our work handles complex scenarios, involving both algebraic loops and step size negotiations. To the best of our knowledge, this is the first approach to do so.

5 CONCLUDING REMARKS

This paper presented an approach for deriving implementation aware co-simulation algorithms for complex co-simulation scenarios where step negotiation and solving algebraic loops are needed as a part of the simulation. The derived algorithms require that the SUs support restoration of state to solve algebraic loops and handle step negotiation. As one of the reviewers pointed out, the methodology presented here can be applied to synthesis of simulation algorithms for Causal Block Diagrams (see, e.g., (Gomes et al. 2020) and references thereof, for an introduction). In particular, different integrator blocks may apply different numerical solvers with different step size requirements, leading to the need to ensure that the simulation algorithm respects these requirements.

Future work includes extending the approach to cover adaptive co-simulations, hierarchical co-simulations, and the generation of parallel co-simulation algorithms. An implementation of the approach and case studies is available online, and the approach currently being implemented as a plugin to Maestro V2.

ACKNOWLEDGMENTS

We are grateful to the Poul Due Jensen Foundation, which has supported the establishment of a new Centre for Digital Twin Technology at Aarhus University. We are grateful to DIGIT for supporting the project both financially and technologically. Finally, we thank the reviewers for their thorough and pertinent comments.

REFERENCES

Arnold, M. 2010, May. "Stability of Sequential Modular Time Integration Methods for Coupled Multibody System Models". *Journal of Computational and Nonlinear Dynamics* vol. 5 (3), pp. 9.

Arnold, M., C. Clauß, and T. Schierz. 2014. "Error Analysis and Error Estimates for Co-Simulation in FMI for Model Exchange and Co-Simulation v2.0". In *Progress in Differential-Algebraic Equations*, pp. 107–125. Berlin, Heidelberg, Springer Berlin Heidelberg.

Benedikt, M., and F. R. Holzinger. 2016, April. "Automated Configuration for Non-Iterative Co-Simulation". In *17th International Conference on Thermal, Mechanical and Multi-Physics Simulation and Experiments in Microelectronics and Microsystems (EuroSimE)*, pp. 1–7. Montpellier, IEEE.

Blockwitz, T., M. Otter, J. Akesson, M. Arnold, C. Clauss, H. Elmqvist, M. Friedrich, A. Junghanns, J. Mauss, D. Neumerkel, H. Olsson, and A. Viel. 2012. "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*, Volume 76, pp. 173–184, Linköping University Electronic Press.

Broman, D., C. Brooks, L. Greenberg, E. A. Lee, M. Masin, S. Tripakis, and M. Wetter. 2013. "Determinate Composition of FMUs for Co-Simulation". In *Eleventh ACM International Conference on Embedded Software*, pp. Article No. 2, IEEE Press Piscataway, NJ, USA.

Busch, M. 2016, September. "Continuous Approximation Techniques for Co-Simulation Methods: Analysis of Numerical Stability and Local Error". *Journal of Applied Mathematics and Mechanics* vol. 96 (9), pp. 1061–1081.

FMI 2014. *Functional Mock-up Interface for Model Exchange and Co-Simulation*. FMI development group.

Galtier, V., S. Vialle, C. Dad, J.-P. Tavella, J.-P. Lam-Yee-Mui, and G. Plessis. 2015. "FMI-based distributed multi-simulation with DACCOSIM". In *SpringSim (TMS-DEVS)*, edited by F. Barros, M. H. Wang, H. Prähofer, and X. H. 0002, pp. 39–46, SCS/ACM.

Gomes, C., D. Broman, H. Vangheluwe, C. Thule, and P. G. Larsen. 2018. "Co-Simulation: A Survey". *ACM Computing Surveys* vol. 51 (3), pp. 49–49.

Gomes, C., J. Denil, and H. Vangheluwe. 2020. "Causal-Block Diagrams: A Family of Languages for Causal Modelling of Cyber-Physical Systems". In *Foundations of Multi-Paradigm Modelling for Cyber-Physical Systems*, edited by P. Carreira, V. Amaral, and H. Vangheluwe, pp. 97–125. Cham, Springer International Publishing.

Gomes, C., L. Lucio, and H. Vangheluwe. 2019. "Semantics of Co-Simulation Algorithms with Simulator Contracts". In *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, pp. 784–789, IEEE.

Gomes, C., B. Meyers, J. Denil, C. Thule, K. Lausdahl, H. Vangheluwe, and P. De Meulenaere. 2019. "Semantic Adaptation for FMI Co-Simulation with Hierarchical Simulators". vol. 95 (3), pp. 1–29.

Gomes, C., B. Oakes, M. Moradi, A. Gámiz, J. Mendo, S. Dutré, J. Denil, and H. Vangheluwe. 2019. "HintCO – Hint-based Configuration of Co-simulations". In *Proceedings of the 9th International Conference on Simulation and Modeling Methodologies, Technologies and Applications*, pp. 57–68, SCITEPRESS - Science and Technology Publications.

Gomes, C., C. Thule, K. Lausdahl, P. G. Larsen, and H. Vangheluwe. 2018. "Stabilization Technique in INTO-CPS". In *2nd Workshop on Formal Co-Simulation of Cyber-Physical Systems*, Volume 11176, Springer, Cham.

Gomes, C., C. Thule, L. Lúcio, H. Vangheluwe, and P. G. Larsen. 2019. "Generation of Co-simulation Algorithms Subject to Simulator Contracts". In *SEFM Workshops*, edited by J. Cámara and M. Steffen, Volume 12226, pp. 34–49, Springer.

Hansen, S. T., C. Thule, and C. Gomes. 2021. "An FMI-Based Initialization Plugin for INTO-CPS Maestro 2". In *Software Engineering and Formal Methods. SEFM 2020 Collocated Workshops*, edited by L. Cleophas and M. Massink, pp. 295–310. Virtual event, Springer International Publishing.

Holzinger, F., and M. Benedikt. 2019a. "Optimal Trigger Sequence for Non-Iterative Co-Simulation:". In *Proceedings of the 9th International Conference on Simulation and Modeling Methodologies, Technologies and Applications*, pp. 80–87. Prague, Czech Republic, SCITEPRESS - Science and Technology Publications.

Holzinger, F. R., and M. Benedikt. 2019b. "Hierarchical Coupling Approach Utilizing Multi-Objective Optimization for Non-Iterative Co-Simulation". In *13th International Modelica Conference 2019*, pp. 6. Regensburg, Germany, Linköping University Electronic Press.

Kalmar-Nagy, T., and I. Stanciulescu. 2014, January. "Can Complex Systems Really Be Simulated?". *Applied Mathematics and Computation* vol. 227, pp. 199–211.

Kübler, R., and W. Schiehlen. 2000. "Two Methods of Simulator Coupling". *Mathematical and Computer Modelling of Dynamical Systems* vol. 6 (2), pp. 93–113.

Oakes, B. J., C. Gomes, F. R. Holzinger, M. Benedikt, J. Denil, and H. Vangheluwe. 2020. "Hint-Based Configuration of Co-Simulations with Algebraic Loops". In *Simulation and Modeling Methodologies, Technologies and Applications*, edited by M. Obaidat, T. Ören, and H. Szczerbicka, Volume 1260, pp. 1–28. Springer International Publishing.

Schweizer, B., P. Li, and D. Lu. 2015. "Explicit and Implicit Cosimulation Methods: Stability and Convergence Analysis for Different Solver Coupling Approaches". *Journal of Computational and Nonlinear Dynamics* vol. 10 (5), pp. 051007. Publisher: ASME.

Tarjan, R. E. 1972. "Depth-First Search and Linear Graph Algorithms". *SIAM J. Comput.* vol. 1 (2), pp. 146–160.

AUTHOR BIOGRAPHIES

SIMON THRANE HANSEN is a Ph.D. student at Aarhus University. He is working in the area of theoretical and applied formal techniques. His email address is sth@ece.au.dk.

CLÁUDIO GOMES is a postdoc researcher at Aarhus University. He received his PhD at the University of Antwerp, for his work on the foundations of co-simulation. His email address is claudio.gomes@ece.au.dk.

JACO VAN DE POL is a professor of Computer Science at Aarhus University. He received his PhD from Utrecht University. His main research interest is model-checking. His email address is jaco@cs.au.dk.

PETER GORM LARSEN is a professor at Aarhus University. He leads the AU DIGIT Centre, the AU Centre for Digital Twins, and the research group for CPSs. His email address is pgl@ece.au.dk.

A ALGORITHMS

Algorithm 1 Step procedure for the scenario in Fig. 1a.

$(s_B^{(s+H)}, H) \leftarrow \texttt{doStep}_B(s_B^{(s)}, H)$
$g_{val} \leftarrow \texttt{get}_B(s_B^{(s+H)}, y_g)$
$s_A^{(s)} \leftarrow \texttt{set}_A(s_A^{(s)}, u_g, g_{val})$
$(s_A^{(s+H)}, H) \leftarrow \texttt{doStep}_A(s_A^{(s)}, H)$
$f_{val} \leftarrow \texttt{get}_A(s_A^{(s+H)}, y_f)$
$s_B^{(s+H)} \leftarrow \texttt{set}_B(s_B^{(s+H)}, u_f, f_{val})$

Algorithm 2 Step procedure for scenario in Fig. 2a.

$(s_{A_v}^{(s)}, s_{B_v}^{(s)}) \leftarrow (s_A^{(s)}, s_B^{(s)})$ ▷ Save A and B
while !*conv* **do** ▷ Fixed-point iteration procedure
  $s_A^{(s)} \leftarrow \texttt{set}_A(s_A^{(s)}, u_f, f_{val})$ ▷ Initialized with Guess
  $s_B^{(s)} \leftarrow \texttt{set}_B(s_B^{(s)}, u_x, x_{val})$ ▷ Initialized with Guess
  $(s_A^{(s+h)}, h), \leftarrow \texttt{doStep}_A(s_A^{(s)}, h)$
  $(s_B^{(s+h)}, h), \leftarrow \texttt{doStep}_B(s_B^{(s)}, h)$
  $x_{vala} \leftarrow \texttt{get}_A(s_A^{(s+h)}, y_x)$
  $f_{vala} \leftarrow \texttt{get}_B(s_B^{(s+h)}, y_f)$
  $conv \leftarrow \mathit{CheckConvergence}((f_a, f_v), (x_a, x_v))$
  if !*conv* **then**
    $s_A^{(s)} \leftarrow s_{A_v}^{(s)}$ ▷ Restore A
    $s_B^{(s)} \leftarrow s_{B_v}^{(s)}$ ▷ Restore B
  end if
  $(f_{val}, x_{val}) \leftarrow (f_{vala}, x_{vala})$ ▷ Update x and f
end while
$(s_C^{(s+h)}, h), \leftarrow \texttt{doStep}_C(s_C^{(s)}, h)$
$j_{val} \leftarrow \texttt{get}_C(s_C^{(s+h)}, y_j)$
$s_B^{(s+h)} \leftarrow \texttt{set}_B(s_B^{(s+h)}, u_j, j_{val})$

Algorithm 3 Step procedure containing fixed-point iteration inside step negotiation procedure for simulating the scenario in Fig. 5a.

SaveSUs in $B \cup K$
$h \leftarrow H_{max}$
while !*Step_found* **do** ▷ Step negotiation
  while !conv **do** ▷ Fixed-point Iteration
    $s_A^{(s)} \leftarrow \texttt{set}_A(s_A^{(s)}, u_f, f_{val})$
    $s_B^{(s)} \leftarrow \texttt{set}_B(s_B^{(s)}, [u_v, u_g], [v_{val}, g_{val}])$ ▷ Setting v and g
    $(s_C^{(s+h_C)}, h_C), \leftarrow \texttt{doStep}_C(s_C^{(s)}, h)$
    $(s_B^{(s+h_B)}, h_B), \leftarrow \texttt{doStep}_B(s_B^{(s)}, h)$
    $(s_A^{(s+h_A)}, h_A), \leftarrow \texttt{doStep}_A(s_A^{(s)}, h)$
    $(v_{vala}, x_{val}) \leftarrow \texttt{get}_A(s_A^{(s+h_A)}, [y_v, y_x])$ ▷ Getting v and x
    $z_{val} \leftarrow \texttt{get}_B(s_B^{(s+h_B)}, y_z)$
    $s_C^{(s+h_C)} \leftarrow \texttt{set}_C(s_C^{(s+h_C)}, u_z, z_{val})$
    $g_{vala} \leftarrow \texttt{get}_C(s_C^{(s+h_C)}, y_g)$
    $s_B^{(s+h_B)} \leftarrow \texttt{set}_B(s_B^{(s+h_B)}, u_x, x_{val})$
    $f_{vala} \leftarrow \texttt{get}_B(s_B^{(s+h_B)}, y_F)$
    $conv \leftarrow \mathit{CheckConvergence}((g_{vala}, g_{val}), (v_{vala}, v_{val}), (f_{vala}, f_{val}))$
    if !*conv* **then**
      RestoreSUs in K
    end if
    $(g_{val}, v_{val}, f_{val}) \leftarrow (g_{vala}, v_{vala}, f_{vala})$
  end while
  $h \leftarrow min(h_A, h_B, h_C)$
  $Step\_found \leftarrow h == h_A \wedge h == h_B \wedge h == h_C$
  if !*Step_found* **then**
    RestoreSUs in B
  end if
end while

Algorithm 4 Step negotiation procedure of scenario in Fig. 4a.

SaveSUs in B
while !*Step_found* **do** ▷ Step negotiation
  $(s_D^{(s+h_D)}, h_D) \leftarrow \texttt{doStep}_D(s_D^{(s)}, h)$
  $g_{val} \leftarrow \texttt{get}_D(s_D^{(s+h_D)}, y_g)$
  $s_C^{(s)} \leftarrow \texttt{set}_C(s_C^{(s)}, u_g, g_{val})$
  $(s_C^{(s+h_C)}, h_C) \leftarrow \texttt{doStep}_C(s_C^{(s)}, h_D)$
  $h \leftarrow min(h_C, h_D)$
  $Step\_found \leftarrow h == h_C \wedge h == h_D$
  if !*Step_found* **then** ▷ Restore SUs in B
    $s_C^{(s)} \leftarrow s_{C_v}^{(s)}$
    $s_D^{(s)} \leftarrow s_{D_v}^{(s)}$
  end if
end while
$z_{val} \leftarrow \texttt{get}_C(s_C^{(s+h_C)}, y_z)$
$s_D^{(s+h_C)} \leftarrow \texttt{set}_D(s_D^{(s+h_D)}, u_z, z_{val})$