Proceedings of the 2020 Winter Simulation Conference
K.-H. Bae, B. Feng, S. Kim, S. Lazarova-Molnar, Z. Zheng, T. Roeder, and R. Thiesing, eds.

ROLLBACK SUPPORT IN HYFLOW MODULAR MODELS

Fernando Barros

Departamento de Engenharia Informática
Universidade de Coimbra
3030 Coimbra, Portugal

ABSTRACT

In this paper we develop an extension of the HYFLOW (Hybrid Flow System Specification) formalism, for providing rollback support to hybrid modular models. HYFLOW extension preserves formalism modularity enabling the co-simulation of hierarchical models. Rollback plays an important role in some numerical methods. Examples include predictor corrector integrators (PCIs), and zero detectors (ZDs). These methods require simulation time to move both forward and backward in order to compute better estimates for the numerical solutions. We present formalism semantics and a description of PCI and ZD methods.

1 INTRODUCTION

Some numerical methods require rollback to compute accurate solutions. Examples include predictor-corrector integrators (PCIs), and zero detectors/(root finding) (ZDs) algorithms. For example, a zero detector (ZD), can use a fixed sampling period while a zero is not found. Upon detecting a zero within a time interval, the sampling needs to be adjusted for accurately find the zero in that interval. This process requires time to move forward and backward so the solution can be found. While finding a zero of a function is a simple task, the problems arise when the ZD is part of a larger model. In this scenario, the algorithm becomes more complex since it needs also to rollback the other models every time the ZD times goes backward. This process becomes more involving when the other models need also to rollback, requiring a sound semantics so ZDs, for example, can be part of a model network. These problems are commonly tackled by modeling tools that handle complex models as non-modular units, requiring the internals of each model to be known (Bouissou and Chapoutot 2012; Fritzson 2004). This approach leads to modeling & simulation tools that can offer modularity at the user level, but internally, these tools perform model transformations that do not enforce the corresponding modular simulation. This solution is effective while models are kept in the same environment. However, they do not guarantee the interoperability of models developed in different tools, since no modular semantics is commonly supported.

Modularity plays an essential role in model interoperability and co-simulation. The ability to combine independent models that are known only by their interface has been subject of simulation standards including the High Level Architecture (HLA) (Kuhl et al. 1999), and the Functional Mock-up Interface (FMI) for Model Exchange and Co-Simulation (Blochwitz et al. 2012).

In order to represent additional numerical methods we exploit here rollback as a key operator to describe models that require the ability to go backward in simulation time with the information gathered in the future. In this paper we present an extension of the HYFLOW formalism (Barros 2003), for supporting rollback while keeping model interoperability and co-simulation.

2 RELATED WORK

Rollback has been used for providing solutions to problems in different domains. Databases have employed this concept to support transactions, where rollback is used for setting the model in a valid state if some operations do not succeeded in a transaction that is intended to be performed atomically (Gray 1978). This concept has also been used in parallel optimistic simulation enabling the model to be set into a valid state when simulation messages arrive out of (time) order (Jefferson 1985). The introduction of rollback in parallel programming was accomplished through the concept of transactional memory, enabling expressing parallelism with the use of lock-free (optimist) operators that use rollback when the optimistic assumptions are not fulfilled (Herlihy and Moss 1993).

Some attempts to describe rollback in modeling did not achieve a sound semantics for enabling the co-simulation of hybrid systems (Lee and Zhen 2005). Rollback has also been used in the context timed automata (Lynch and Vaandrager 1996). However, to the best of our knowledge, the support for rollback in the context of hybrid hierarchical, modular models is currently missing.

The representation of hybrid systems has been subjected to extensive research. The DEV&DESS (Praehofer 1991), for example, belongs to a common family of representations that abstracts continuous signals as exact. Since these types of formalisms do not provide an explicit description of numerical integrators they cannot be used to model systems where certain proprieties need to be guaranteed. This is the case, for example, of energy conservation systems that require the use of geometric integrators (Hairer et al. 2005), to achieve accurate simulation results in large time intervals. Given HYFLOW ability to describe numerical methods it can be used to make the interoperability of different numerical integrators (Barros 2018). Many simulation languages like Modelica (Fritzson 2004), do not support co-simulation. They usually convert an *apparently modular* model into a large atomic model that is simulated in a non-modular form. HYFLOW, on the contrary, enables independent modular models to be co-simulated.We note that, given HYFLOW modular design, a user not requiring, for example, new types of integrators can build new models just by assembling existing components. However, HYFLOW enables users to develop new algorithms, like integrators or zero-cross detectors, in a modular form. This kind of flexibility is not enabled by other approaches like DEVS&DESS and Modelica.

There are currently different definitions of hybrid simulation. In the operational research area, for example, hybrid simulation is considered as a combination of at least two of the following paradigms: discrete-event (DE), system dynamics (SDs), and agent-based simulation (ABS) (Brailsford et al. 2019). HYFLOW follows a different perspective based on the semantics of *hybrid flow* trajectories. This semantics enables the creation of basic modular models that can be used to describe more complex systems. For example, the numerical integrator defined in Section 4 can be used as the basic block to describe SD models. In a similar manner other paradigms, like fluid stochastic Petri-nets, can be described in HYFLOW (Barros 2015). The extension of HYFLOW with the rollback operator enables models to use non-causal numerical methods to achieve more accurate simulations. This is the case, for example, of the detector described in Section 5, that can signal a zero-cross event with an arbitrary accuracy.

In the next section we present the HYFLOW and its semantics, showing formalism ability to compose and co-simulate modular hybrids models with rollback.

3 THE HYFLOW FORMALISM

The Hybrid Flow System Specification (HYFLOW) is a formalism aimed to represent hybrid systems with a time-variant topology (Barros 2003). HYFLOW achieves the representation of continuous variables using the concepts of multi-sampling and dense outputs (Barros 2002), while the representation of discrete events is based on the Discrete Event System Specification (DEVS) (Zeigler 1984). HYFLOW has two types of models: basic and network. Basic models provide state representation and transition functions. Network models are a composition of basic models and other network models. Given its definition, a network provides an abstraction for representing hierarchical systems.

3.1 Basic Model

We consider $\widehat{B}$ as the set of names corresponding to basic HYFLOW models. A HYFLOW basic model associated with name $B \in \widehat{B}$ is defined by:

$$M_B = (X, Y, P, P_0, \rho, \omega, \delta, \Lambda_c, \lambda_d)$$

where

- $X = X_c \times X_d$ is the set of input flow values
  - X_c is the set of continuous input flow values
  - X_d is the set of discrete input flow values
- $Y = Y_c \times Y_d$ is the set of output flow values
  - Y_c is the set of continuous output flow values
  - Y_d is the set of discrete output flow values
- P is the set of partial states (p-states)
- $P_0 \subseteq P$ is the set of (valid) initial p-states
- $\rho : P \longrightarrow \mathbb{H}$ is the time-to-input function
- $\omega : P \longrightarrow \mathbb{H}$ is the time-to-output function
- $S = \{(p,e) | p \in P, 0 \leq e \leq \nu(p)\}$ is the state set, with
  - $\nu(p) = \min\{\rho(p), \omega(p)\}$, the time-to-transition function
- $\delta : S \times X^{\phi} \times P^{\phi} \longrightarrow P$ is the transition function, with
  - $X^{\phi} = X_c \times (X_d \cup \{\phi\})$,
  - $P^{\phi} = P \cup \{\phi\}$,
  - and ϕ is the null value (absence of value)
- $\Lambda_c : S \longrightarrow Y_c$ is the continuous output function
- $\lambda_d : P \longrightarrow Y_d$ is the partial discrete output function

The HYFLOW time base is the set of hyperreals numbers $\mathbb{H} = \{x + z\varepsilon | x \in \mathbb{R}, z \in \mathbb{Z}\}$, where ε is an infinitesimal value, such that $\varepsilon > 0$ and $\varepsilon < \frac{1}{n}$ for $n = 1, 2, 3, ...$ (Goldblatt 1998). The discrete output of a component described by a HYFLOW basic model is constrained to be null (ϕ) when in states (s,e) with $e \neq \omega(p)$.

Time functions include negative values to define rollback interval. When the model rollbacks, the transition function receives the p-state at rollback time. This p-state enables the model to make a decision based on information brought from the future.

Figure 1 depicts the typical trajectories of a HYFLOW component. At time t_1 the component in p-state p_0 samples its input, since its elapsed time reaches $\rho(p_0) = e_0$. The component changes its p-state to $p_1 = \delta((p_0, \rho(p_0)), (x_1, \phi), \phi)$, where x_1 is the sampled value and no discrete flow is present. At time t_2 the discrete flow x_d is received by the component that changes to p-state $p_2 = \delta((p_1, e_1), (x_2, x_d), \phi)$, where x_2 is the continuous flow at t_2. At time t_3 the component reaches the time-to-output time limit and it changes to p-state $p_3 = \delta((p_2, \omega(p_2)), (x_3, \phi), \phi)$. At this time the discrete flow $y_d = \lambda_d(p_2)$ is produced. Additionally, component continuous output flow is always present and given by $\Lambda_c(p,e)$. The component can also define a rollback by specifying a negative value for time-to-input/output functions.

Considering that $\rho(p_3) < 0$, the model rolls back to time $t'_3 = t_3 - \rho(p_3)$. Assuming $t'_3 < t_2$, the component changes to p-state $\delta((p_1, t'_3 - t_1), (x'_3, \phi), p_3)$. This transition occurs backward at time t'_3 with p-state p_3 gathered at future time $t_3 > t'_3$. A more detailed description of formalism semantics is provided in the next sections through the concept of HYFLOW modular components.

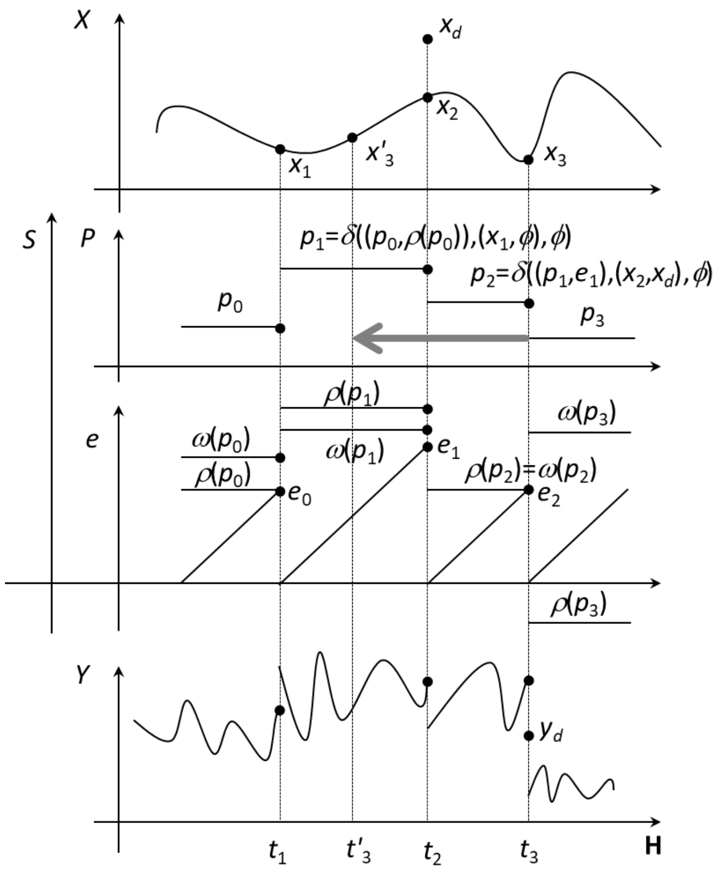

Figure 1: Basic HYFLOW component trajectories.

3.2 Basic Component

To describe HYFLOW semantics we extend here the concept of component defined in Barros (2008) with the rollback mechanism. Components have their behavior governed by HYFLOW models but they introduce the rules on how models are interpreted. We consider that a basic component associated with name $B \in \widehat{B}$ and model $M_B = (X,Y,P,P_0,\rho,\omega,\delta,\Lambda_c,\lambda)$, has two state variables: $s \in S$, and $\varphi \in P^{\phi}$; where S is the component transition history set, and $\varphi \in P^{\phi}$ is the component future p-state. Given HYFLOW non-instantaneous assumption (Barros 2008), an event $(t+\varepsilon,p) \in E$ describes the effect of a transition triggered at time t that has changed component current p-state to p. Component transition history keeps record of past transitions, enabling rollback, with the information $\varphi \in P^{\phi}$ gathered at time in the future. Component transition history set is defined by:

$$S = \mathcal{P}(E)$$

where

$E = \{(t,p) | t \in \mathbb{H} \wedge p \in P\}$ is the set of events

$\mathcal{P}(E)$ is the power set of E

To simplify the description of component semantics, we introduce the concept of *action*, defined here as an extend function that can read/modify component state variables. A basic component associated with name $B \in \widehat{B}$ and model $M_B = (X,Y,P,P_0,\rho,\omega,\delta,\Lambda_c,\lambda)$ is defined by:

$$\Xi_B = (\mathrm{N},\Lambda,\Delta,R)$$

where

```
N :: $\longrightarrow \mathbb{H}$, is the component next time transition action, defined by:
   $N() \triangleq$
      $(t_k, p_k) \leftarrow \max_t\{(t,p) \in s\}$
      return $t_k + \nu(p_k)$
$\Lambda :: \mathbb{H} \longrightarrow Y^{\phi}$, is the component output action, defined by:
   $\Lambda(t) \triangleq$
      $(t_k, p_k) \leftarrow \max_t\{(t,p) \in s\}$
      if $(t - t_k = \omega(p_k))$
         return $(\Lambda_c(p_k, t - t_k), \lambda(p_k))$
      else
         return $(\Lambda_c(p_k, t - t_k), \phi)$
   with $Y^{\phi} = Y_c \times (Y_d \cup \{\phi\})$
$\Delta :: \mathbb{H} \times X^{\phi}$, is the component transition action, defined by:
   $\Delta(t, (x_c, x_d)) \triangleq$
      if $(t \neq N() \wedge x_d = \phi \wedge \varphi = \phi)$ return
      $(t_k,\ p_k) \leftarrow \max_t\{(t,p) \in s\}$
      $\varphi \leftarrow \delta((p_k,\ t - t_k), (x_c,\ x_d), \varphi)$
      if $(\nu(\varphi) \geq 0)$ {
         $s \leftarrow s \cup \{(t + \varepsilon, \varphi)\}$
         $\varphi \leftarrow \phi$
      }
```

Given a transition at time t the component p-state, by the non-instantaneous propagation, changes only at $t + \varepsilon$.

```
$R :: \mathbb{H}$, is the component rollback action, defined by:
   $R(t) \triangleq$
      $s \leftarrow \{(t_k, p_k) \in s \wedge t_k < t\}$
      if $(s = \{\})$ error
```

Given two numbers $x, y \in \mathbb{H}$, with $x = a + m\varepsilon$ and $y = b + n\varepsilon$, the relational operator $<$ is defined by:

$$x < y \quad \text{if } (a < b) \vee (a = b \wedge m < n)$$

Simulation is performed by components that have their behavior governed by the associated model. Component simulation algorithm is described in Section 3.6, that employs component actions to run simulations. Component self-scheduling transition time is given by action N. Component output is given by action Λ. The Δ action describes component state after a transition occurs. This action receives as parameter the state gather in a future time, when the rollback was requested by the component. Component action R is responsible to update component state when time is rolled back.

3.3 Network Model

HYFLOW networks are compositions of HYFLOW models (basic or other HYFLOW network models). Additionally, each network has a special component, named as the executive, that is responsible for defining network topology (composition and coupling). Let $\widehat{N}$ be the set of names corresponding to HYFLOW network models, with $\widehat{N} \cap \widehat{B} = \{\}$. A HYFLOW network model associated with name N is defined by:

$$M_N = (X, Y, \eta)$$

where

N is the network name

$X = X_c \times X_d$ is the set of network input flows

X_c is the set of network continuous input flows

X_d is the set of network discrete input flows

$Y = Y_c \times Y_d$ is the set of network output flows

Y_c is the set of network continuous output flows

Y_d is the set of network discrete output flows

η is the name of the dynamic topology network executive

The executive is a HYFLOW basic model extended with topology related operators. The executive model is defined by:

$$M_\eta = (X_\eta, Y_\eta, P, P_0, \rho, \omega, \delta, \Lambda_c, \lambda_d, \widehat{\Sigma}, \gamma)$$

where

$\widehat{\Sigma}$ is the set of network topologies

$\gamma : P \longrightarrow \widehat{\Sigma}$ is the topology function

The network topology $\Sigma_\alpha \in \widehat{\Sigma}$, corresponding to the p-state $p_\alpha \in P$, is given by:

$$\Sigma_\alpha = \gamma(p_\alpha) = (C_\alpha, \{I_{i,\alpha}\} \cup \{I_{\eta,\alpha}, I_{N,\alpha}\}, \{F_{i,\alpha}\} \cup \{F_{\eta,\alpha}, F_{N,\alpha}\})$$

where

C_α is the set of names associated with the executive state p_α

for all $i \in C_\alpha \cup \{\eta\}$

$I_{i,\alpha}$ is the sequence of influencers of i

$F_{i,\alpha}$ is the input function of i

$I_{N,\alpha}$ is the sequence of network influencers

$F_{N,\alpha}$ is the network output function

For all $i \in C_\alpha$

$M_i = (X, Y, P, P_0, \rho, \omega, \delta, \Lambda_c, \lambda_d)_i$ if $i \in \widehat{B}$

$M_i = (X, Y, \eta)_i$ if $i \in \widehat{N}$

The topology of a network is defined by its executive through the topology function γ, which maps the executive p-state into network composition and coupling. Thus, topology adaption can be achieved by changing the executive p-state. HYFLOW network models are simulated by HYFLOW network components that perform the orchestration of basic or other networks. Network co-simulation is achieved by a general communication protocol that relies only on the component interface. This protocol is independent from model details, enabling the composition of components that are interpreted as black boxes.

3.4 Executive Component

Before describing the network component we define the executive component, an extension to the basic component, that introduces the topology function required to establish network topology. A HYFLOW executive component Ξ_η with $\eta \in \widehat{\eta}$, associated with the executive model $M_\eta = (X,Y,P,P_0,\rho,\omega,\delta,\Lambda_c,\lambda,\widehat{\Sigma},\gamma)$ is defined by:

$$\Xi_\eta = (\mathrm{N},\Lambda,\Delta,R,\Gamma)$$

where

```
$\Gamma :: \longrightarrow \widehat{\Sigma}$, is the executive component topology action defined by:
  $\Gamma() \triangleq$
    $(t_k, p_k) \leftarrow \max_t \{(t,p) \in s\}$
    return $\gamma(p_k)$
```

Executive component action Γ defines the current network topology and is used in the next section to define network semantics and in particular network dynamic topology.

3.5 Network Component

A network is composed by the executive component and a set of other components. These components and their interconnections can change according to the current state of the executive. Components can be basic or other HYFLOW networks, making possible to define networks hierarchically. A HYFLOW network component Ξ_N with $N \in \widehat{N}$, associated with the network model $M_N = (X,Y,\eta)$, executive $M_\eta = (X_\eta,Y_\eta,P,P_0,\rho,\omega,\delta,\Lambda_c,\lambda,\widehat{\Sigma},\gamma)$, and current topology $\Gamma_\eta() = (C,\{I_i\} \cup \{I_\eta,I_N\},F_i \cup \{F_\eta,F_N\})$, is defined by:

$$\Xi_N = (\mathrm{N},\Lambda,\Delta,R)$$

where

```
$\mathrm{N} :: \longrightarrow \mathbb{H}$, is the time of next transition, defined by:
  $\mathrm{N}() \triangleq$ return $\min_{k \in C \cup \{\eta\}} \{\mathrm{N}_k()\}$
$\Lambda :: \mathbb{H} \longrightarrow Y^\phi$, is the network component output, defined by:
  $\Lambda(t) \triangleq$ return $F_N(\bigtimes_{k \in I_N} \Lambda_k(t))$
$\Delta :: \mathbb{H} \times X^\phi$, is the network component transition, defined by:
  $\Delta(t,x) \triangleq$
    $y[N] \leftarrow x$
    $\forall_{k \in C}\; y[k] \leftarrow \Lambda_k(t)$
    $\forall_{k \in C}\; \Delta_k(t, F_k(\bigtimes_{j \in I_k} y[j]))$
    $\Delta_\eta(t, F_\eta(\bigtimes_{j \in I_\eta} y[j]))$
$R :: \mathbb{H}$, is the component rollback action, defined by:
  $R(t) \triangleq$
    $\forall_{k \in C}\; R_k(t)$
    $R_\eta(t)$
```

In the rollback action, determinism is guaranteed by making the executive the last component that undergoes rollback.

3.6 Component Simulation

Component simulation is performed by the action:

$$\mathcal{S} :: \mathfrak{M} \times \mathbb{H}$$

where $\mathfrak{M} = \{m \,|\, M_m = (\{\} \times X_d, Y, \ldots), m \in \widehat{B} \cup \widehat{N}\}$, is the set of names associated with HFSS models (basic or network) defining a null continuous input flow interface. The simulation action is defined by:

```
$\mathcal{S}(m, \Omega) \triangleq$
    $\mathfrak{C} \leftarrow \mathrm{N}_m()$
    while $(\mathfrak{C} < \Omega)$ {
        $\Delta_m(\mathfrak{C}, (\phi, \phi))$
        $\mathfrak{c} \leftarrow \mathrm{N}_m()$
        while $(\mathfrak{c} < \mathfrak{C})$ {
            $R_m(\mathfrak{c})$
            $\mathfrak{C} \leftarrow \mathfrak{c}$
            $\Delta_m(\mathfrak{C}, (\phi, \phi))$
        }
        $\mathfrak{C} \leftarrow \mathfrak{c}$
    }
```

Simulation, in addition to supporting the regular forward time behavior, also needs to enable consecutive rollbacks to occur, as enabled by action inner loop. In the next section we present a predictor-corrector integrator, that provides an application example for HYFLOW rollback semantics.

4 PREDICTOR-CORRECTOR INTEGRATOR

Many numerical methods, like integrators, require rollback, since they use look-ahead into the future in order to make a better estimator of that future. A simple example is the predictor corrector integrator (PCI). A simple algorithm uses the Euler method to make the first prediction, and then it can use the trapezoidal rule to increase the accuracy. The PCI can be described by the following equations:

$$\begin{cases} y^p_{k+1} = y_k + hf(x_k, y_k) \\ y^c_{k+1} = y_k + \frac{1}{2}h[f(x_k, y_k) + f(x^p_{k+1}, y^p_{k+1})] \end{cases}$$

When used in isolation PCIs can be implemented in a straightforward manner. However, the main problem is model interoperability when, for example, we need to combine the integrator with a zero detection algorithm. In this case the integrator output can no longer be described by a discrete set of values, but a dense output representation is required. Moreover, since the integrator rollback, any component attached to it needs also to rollback so all components can be synchronized. The HYFLOW model of the PCI with a constant step-size $h > 0$ can be defined by:

$$M_{PCI} = (X, Y, P, P_0, \rho, \omega, \delta, \Lambda_c, \lambda_d)$$

where

$$
\begin{aligned}
&X = \mathbb{R} \times \{\} \\
&Y = \mathbb{R} \times \{\} \\
&P = \{(\varphi, f, f_k, y_k) |\, \varphi \in \{I, A, B, C\}; f \in \mathbb{R}^2 \longrightarrow \mathbb{R}; f_k, y_k \in \mathbb{R}\} \\
&P_0 = \{(\varphi, f, f_k, y_k) |\, \varphi = I\} \\
&\rho(\varphi, f, f_k, y_k) = \begin{cases} 0 & \text{if } \varphi = I \\ -h & \text{if } \varphi = B \\ h & \text{otherwise} \end{cases} \\
&\omega(\varphi, f, f_k, y_k) = \infty \\
&\delta((I, f, f_k, y_k), e, (x_c, x_d), \phi) = (A, f, f(x_c, y_k), y_k) \\
&\delta((A, f, f_k, y_k), e, (x_c, x_d), \phi) = (B, f, f_k, y_k + \text{std}(e) f_k) \\
&\delta((B, f, f_k, y_k), e, (x_c, x_d), \phi) = (C, f, \tfrac{1}{2}(f(x_c, y_{k+1}) + f_k), y_{k+1}), \text{ with } y_{k+1} = y_k + \text{std}(e) f_k \\
&\delta((B, f, f_k, y_k), e, (x_c, x_d), (\varphi, f, f_k, y_k)') = (\varphi', f, f'_k, y_k)) \\
&\delta((C, f, f_k, y_k), e, (x_c, x_d), \phi) = (A, f, f(x_c, y_k), y_k + \text{std}(e) f_k) \\
&\Lambda_c((\varphi, f, f_k, y_k), e) = y_k + \text{std}(e) f_k \\
&\lambda_d(\varphi, f, f_k, y_k) = \phi
\end{aligned}
$$

The PCI phase diagram is depicted in Figure 2. In the initial phase I component samples its input in order to compute the derivative. The component then cycles between phases $A, B,$ and C. In phase A the PCI is in predictor mode. It makes a forward step h, enters phase B and gathers the new derivative. In phase B it moves backward to phase C. With current and predicted derivatives the component moves to phase A with an improved estimator for the solution.

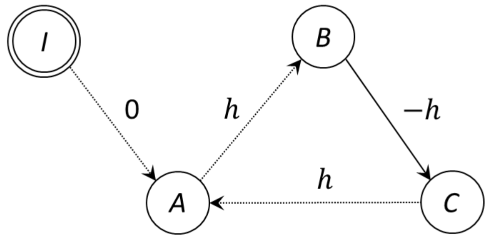

Figure 2: Predictor-corrector integrator phase transitions.

In the next section we describe a zero-detector model that also uses rollback to accurately find the zero values of a continuous flow.

5 ZERO DETECTOR

Zero detectors (ZDs) play a fundamental role in hybrid systems, where they are responsible to monitor continuous variables and detect events that can create discontinues. Although finding a root of a function can be performed by very efficient methods (Stoer and Bulirsh 2002), these methods rely on rollback and they cannot be easily applied in a modular simulation (co-simulation) context. We introduce next a HYFLOW modular description of the false position method (FPM) for zero-detection. FPM is described in Figure 3 that describes method behavior when a zero is detected in the interval $[a, b]$ since $f(a)f(b) < 0$. FMP performs a linear interpolation to find an approximation z, and a new search interval $[a, z]$ is used in the next iteration. The method is repeated until the desired accuracy is achieved. Although the method is quite simple it requires rollback, and a modular representation for this mechanism, as supported by HYFLOW. A zero-detector HYFLOW model can be defined by:

$$M_{ZD} = (X, Y, P, P_0, \rho, \omega, \delta, \Lambda_c, \lambda_d)$$

where

$X = \mathbb{R} \times \{\}$
$Y = \{\} \times \mathbb{R}$
$P = \{(\varphi, \alpha, \beta, a, f_a) | \, \varphi \in \{I, K, F, Z\}; \, \alpha, \beta \in \mathbb{H}; \, a, f_a \in \mathbb{R}\}$
$P_0 = \{(\varphi, \alpha, \beta, a, f_a) | \, \varphi = I, \alpha = 0, \beta = \infty, \, a = 0\}$
$\rho(\varphi, \alpha, \beta, a, f_a) = \alpha$
$\omega(\varphi, \alpha, \beta, a, f_a) = \beta$
$\delta((Z, \alpha, \beta, a, f_a), e, (x_c, x_d), p) = (K, h, \infty, a + e, 0)$
$\delta((\varphi, \alpha, \beta, a, f_a), e, (x_c, x_d), p) = (Z, \infty, 0, a + e, 0)$ if $\varphi \neq Z \wedge |x_c| < tol$
$\delta((\varphi, \alpha, \beta, a, f_a), e, (x_c, x_d), p) = (F, -x_c e/(x_c - f_a), \beta, a + e, x_c)$ if $f_a x_c < 0$
$\delta((\varphi, \alpha, \beta, a, f_a), e, (x_c, x_d), (\varphi, \alpha, \beta, a, f_a)') = (F, -x_c(a' - a - e)/(f_a' - f_a), \beta, a + e, x_c)$ if $f_a x_c \geq 0$
$\delta((I, \alpha, \beta, a, f_a), e, (x_c, x_d), \phi) = (K, h, \infty, a + e, x_c)$
$\delta((K, \alpha, \beta, a, f_a), e, (x_c, x_d), \phi) = (K, h, \infty, a + e, x_c)$ if $f_a x_c \geq 0$
$\Lambda_c(p, e) = \phi$
$\lambda_d(p) = 0$

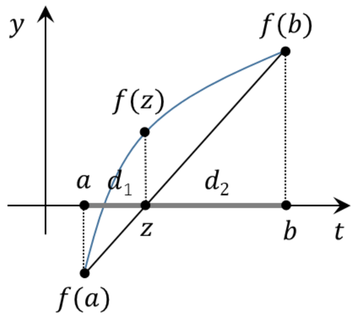

Figure 3: False position method for zero detection.

For simplifying description, the ZD samples its input at a fixed interval h. Upon detection of a zero interval, the FPM is applied to find the zero with an accuracy tol. When the zero is found a discrete output is sent to inform other models of the event. A more robust detection algorithm can use adaptive sampling for ensuring that the function has only one zero within a sampling interval. More efficient algorithms, like Newton-Raphson and fixed point methods (Stoer and Bulirsh 2002), can also be described, since they have similar rollback requirements. In the next section we present simulation results for PCI and ZD models.

6 SIMULATION RESULTS

We consider results for the predictor-corrector integrator of Section 4. HYFLOW network model is described in Figure 4, where G is a function generator with output $f(x) = e^{-x} \cos^2(x) x - 0.1$, I is the PCI, and the input function F_I is the identity. Simulation results are shown in Figure 5, for a step-size of 0.1s. Results are produced with the HYFLOW modeling environment implemented in the TypeScript language (Vanderkam 2019). The ability to combine rollback-based components is demonstrated by the HYFLOW network of

Figure 6, where Z is a zero-detector described in Section 5, and input functions F_I and F_Z are the identity function. The ZD has a sampling interval of 0.02s, while PCI step-size was 0.1s, as before. Zero was detected at times: 0s, 0.23969s, 1.99249s, 2.61746s and 4.13494s, as shown in Figure 5, by looking the zeros of the integral function I$[f(x)]$.

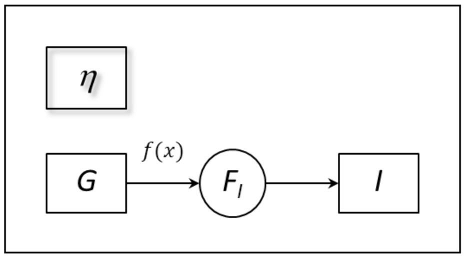

Figure 4: Function generator and PCI network.

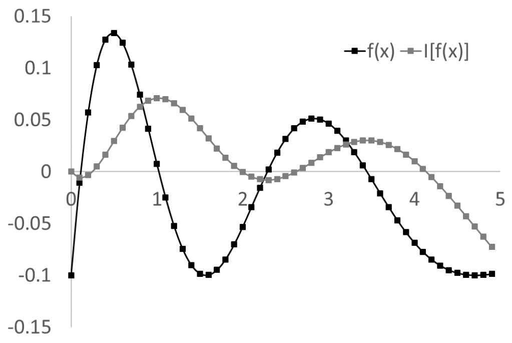

Figure 5: Fixed step-size predictor-corrector integrator.

Given that both components can rollback, the ZD can detect zeros in the predictor and corrector phases of the PCI, a simple optimization strategy would be to stop the ZD during the predictor phase of the integrator.

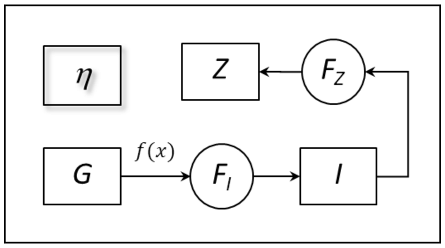

Figure 6: Function generator, PCI, and ZD network.

An interesting case not supported by the current TypeScript implementation is the ability to rollback network dynamic topologies. An efficient storage of the executive component is required to handle these types of models.

CONCLUSION

An extension to the HYFLOW formalism able to represent rollback was presented. Formalism semantics was described using the concept of modular component. The extended HYFLOW keeps the modularity of the original formalism, enabling the composition and co-simulation of modular hybrid models supporting rollback. We have shown that the HYFLOW rollback operator can be used to represent predictor-corrector integrators, and zero detectors based on the false position method. Programming techniques, like reverse computation (Biswas and Mall 1999; Carothers et al. 1999), will be considered to improve the efficiency of rollback. This is particularly relevant for enabling dynamic topology models, a feature not currently supported by the HYFLOW implementation developed in the TypeScript language.

References

Barros, F. 2002. "Towards a Theory of Continuous Flow Models". *International Journal of General Systems* 31(1):29–39.

Barros, F. 2003. "Dynamic Structure Multiparadigm Modeling and Simulation". *ACM Transactions on Modeling and Computer Simulation* 13(3):259–275.

Barros, F. 2008. "Semantics of Dynamic Structure Event-based Systems". In *Proceedings of the 2th International Conference on Distributed Event-based Systems*. July 1st-4th, Rome, Italy, 245-252.

Barros, F. 2015. "A Modular Representation of Fluid Stochastic Petri Nets". In *Proceedings of the Symposium on Theory of Modeling and Simulation*. April 12th-15th, Alexandria, Virginia.

Barros, F. 2018. "Composition of Numerical Integrators in the HyFlow Formalism". In *Proceedings of the Winter Simulation Conference*, edited by M. Rabe, A. Juan, N. Mustafee, A. Skoogh, S. Jain, and B. Johansson. December 9th-12th, Gothenburg, Sweden.

Biswas, B., and R. Mall. 1999. "Reverse Execution of Programs". *ACM SIGPLAN Notices* 34(4):61–69.

Blochwitz, 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. "The Functional Mockup Interface 2.0: The Standard for Tool independent Exchange of Simulation Models". In *Proceedings of the 9th Modelica Conference*. September 3rd-5th, Munich, Germany.

Bouissou, O., and A. Chapoutot. 2012. "An Operational Semantics for Simulink's Simulation Engine". *ACM SIGPLAN Notices* 47(5):129–138.

Brailsford, S., T. Eldabi, M. Kunc, N. Mustafee, and A. Osorio. 2019. "Hybrid Simulation Modelling in Operational Research: A State-of-the-art Review". *European Journal of Operational Research* 278(3):721–737.

Carothers, C., K. Perumalla, and R. Fujimoto. 1999. "Efficient Optimistic Parallel Simulations using Reverse Computation". *ACM Transactions on Modeling and Computer Simulation* 9(3):224–253.

Fritzson, P. 2004. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc. Press.

Goldblatt, R. 1998. *Lectures on the Hyperreals: An Introduction to Nonstandard Analysis*. New York: Springer.

Gray, J. 1978. "Notes on Database Operating Systems". In *Operating Systems. Lecture Notes in Computer Science*, edited by R. Bayer, G. Graham, and G. Seegmüller, Volume 60, 393–481. Berlin: Springer.

Hairer, E., C. Lubich, and G. Wanner. 2005. *Geometrical Numerical Integration: Structure-Preserving Algorithms for Ordinary Differential Equations*. 2nd ed. Number 31 in Springer Series in Computational Mathematics. Berlin: Springer-Verlag.

Herlihy, M., and J. Moss. 1993. "Transactional Memory: Architectural Support for Lock-Free Data Structures". In *Proceedings of the 20th International Symposium on Computer Architecture*. San Diego, California, 289-300.

Jefferson, D. 1985. "Virtual Time". *ACM Transactions on Programming Languages and Systems* 7(3):404–425.

Kuhl, F., R. Weatherly, and J. Dahmann. 1999. *Creating Computer Simulation Systems: An Introduction to the High Level Architecture*. Upper Saddle River, New Jersey: Prentice Hall.

Lee, E., and H. Zhen. 2005. "Operational Semantics of Hybrid Systems". In *Proceedings of the 8th International Workshop on Hybrid Systems: Computation and Control*. March 9th-11th, Zurich, Switzerland, 25-53.

Lynch, N., and F. Vaandrager. 1996. "Forward and Backward Simulations: II. Timing-Based Systems". *Information and computation* 128(1):1–25.

Praehofer, H. 1991. *System Theoretic Foundations for Combined Discrete-Continuous System Simulation*. Ph. D. thesis, University of Linz.

Stoer, J., and R. Bulirsh. 2002. *Introduction to Numerical Analysis*. 3rd ed. Berlin: Springer.

Vanderkam, D. 2019. *Effective TypeScript*. Sebastopol, California: O'Reilly.

Zeigler, B. 1984. *Multifaceted Modelling and Discrete Event Simulation*. San Diego: Academic Press.

AUTHOR BIOGRAPHY

FERNANDO BARROS is a professor at the University of Coimbra, Portugal. His research interests include theory of modeling & simulation and hybrid dynamic topology systems. He published more than 70 papers in journals, book chapters and conference proceedings, and he has organized several conferences and workshops in the area of simulation. Fernando Barros is a member of the editorial board of the *Int. J. Simulation and Process Modelling* and vice-president of the *The Society for Modeling & Simulation International* (SCS). His e-mail address is barros@dei.uc.pt.