Declaration of Unknowns in DAE-Based Hybrid System Specification

D.A. VAN BEEK
Eindhoven University of Technology
V. BOS
Turku Center for Computer Science

and
J.E. ROODA
Eindhoven University of Technology

The majority of hybrid languages is based on the assumption that discontinuities in differential variables at discrete events are modeled by explicit mappings. When there are algebraic equations restricting the allowed new values of the differential variables, explicit remapping of differential variables forces the modeler to solve the algebraic equations. To overcome this difficulty, hybrid languages use many different language elements. This paper shows that only one language element is needed for this purpose: an unknown declaration, that allows the explicit declaration of a variable as unknown. The syntax and semantics of unknown declarations are discussed. Examples are given, using the Chi language, in which unknown declarations are used for modeling multi-body collision, steady-state initialization, and consistent initialization of higher index systems. It is also illustrated how the declaration of unknowns can help to clarify the structure of the system of equations, and how it can help the modeler detect structurally singular systems of equations.

Categories and Subject Descriptors: I.6.2 [**Simulation and Modeling**]: Simulation Languages; I.6.8 [**Simulation and Modeling**]: Types of Simulation—*Combined*; I.6.5 [**Simulation and Modeling**]: Model Development—*Modeling methodologies*

General Terms: Languages
Additional Key Words and Phrases: Consistent initial conditions, initial value problem, hybrid systems, semantics

1. INTRODUCTION

Hybrid languages, generally have the following common characteristics. There is a distinction between 'discrete' and 'continuous' variables. When differential algebraic equations (DAEs) are used, the continuous variables can be divided into

Author's addresses: D.A. van Beek and J.E. Rooda, Department of Mechanical Engineering, Eindhoven University of Technology, POB 513, 5600 MB Eindhoven, Netherlands, e-mail:{d.a.v.beek, j.e.rooda}@tue.nl; V. Bos, Turku Center for Computer Science, DataCity, Lemminkaisenkatu 14 A, 20520 Turku, Finland, e-mail: vbos@abo.fi.

differential variables (or state variables), that occur differentiated in the equations, and algebraic variables, that do not occur differentiated in the equations.

A run of a hybrid model is a alternating sequence of discrete and continuous phases. In a discrete phase, model time is constant and statements are executed (or actions/events take place); the values of the discrete variables and the differential variables can be changed by means of assignments or mappings. In a continuous phase, model time advances, and the values of the continues variables are determined by the equations as a function of time. When a time-event or state-event occurs, the continuous phase terminates, and a discrete phase starts.

Changes in the values of the differential variables in a discrete phase are usually modeled by means of explicit mappings. This is, however, severely complicated when there are (temporary) algebraic equations restricting the allowed new values of the differential variables. There are three types of such algebraic restrictions:

(1) Dependencies between the values of differential variables immediately before and after a discrete event. Such dependencies can be a result of modeling physical phenomena using parameter or time scale abstractions [Mosterman and Biswas 1997; 2002]. An example is a multi-body collision that is modeled to take place at a specific time-point (see Section 6.1).
(2) Additional instantaneous equations that limit the number of dynamic degrees of freedom (see Section 2.1) of the equation system. This is the case for steady state initialization, where $\dot{\mathbf{x}} = \mathbf{0}$. Instantaneous equations are equations that are valid only at certain discrete-event time points (see Section 6.2).
(3) Dependencies between differential variables $\mathbf{x}$. Such dependencies may be in the form of $\mathbf{g}(\mathbf{x}) = \mathbf{0}$, or in the somewhat more general form of: $\mathbf{g}(\mathbf{x}, \mathbf{y}) = \mathbf{0}$, such that $\partial\mathbf{g}/\partial\mathbf{y}$ is singular, where $\mathbf{y}$ denotes the algebraic variables. This is the case in higher index systems (see Section 7).

Many different language elements are used in hybrid languages to deal with the different algebraic restrictions on differential variables. This paper aims to show that, in addition to conditional and instantaneous equations, only one language element is needed for this purpose. The proposed language element is an unknown declaration, that allows the explicit declaration of a variable as unknown.

The paper continues with an introduction to DAEs. The solution of a DAE system is divided into the consistent initialization problem and the initial value problem, which are defined in terms of the equations, unknowns, and knowns. Then, hybrid languages are discussed. The syntax and semantics of the proposed unknown declarations are treated in a general way, and more specifically for the χ (or Chi) language. Subsequently, the use of unknown declarations is illustrated for modeling of colliding bodies, and for steady-state initialization. Finally, the use of unknown declarations in the context of higher index systems is discussed. A PD controller and the well known pendulum are treated as examples.

2. BACKGROUND ON DAES

Consider the following general representation of a DAE that is considered on an interval starting at $t = t_0$

$$\mathbf{f}(\dot{\mathbf{x}}(t), \mathbf{x}(t), \mathbf{y}(t), t) = \mathbf{0}, \tag{1}$$

where $\dot{\mathbf{x}} : \mathbb{R} \rightarrow \mathbb{R}^n$ denotes the n derivatives $\dot{x}_0 \ldots \dot{x}_{n-1}$; $\mathbf{x} : \mathbb{R} \rightarrow \mathbb{R}^n$ denotes the n differential variables $x_0 \ldots x_{n-1}$; $\mathbf{y} : \mathbb{R} \rightarrow \mathbb{R}^m$ denotes the m algebraic variables $y_0 \ldots y_{m-1}$; $t \in \mathbb{R}$ denotes the time, and $\mathbf{f} : \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R} \rightarrow \mathbb{R}^{n+m}$ denotes the set of $n + m$ equations. Mathematically, the solution of DAEs can be divided into two sub-problems that are solved sequentially: the consistent initialization problem [Pantelides 1988], which aims at solving the equations at a certain point of time $t = t_0$; and the initial value problem, which aims at solving the equations for $t \geq t_0$, using the solution of the consistent initialization problem as the starting point [Brown et al. 1998]. In this paper, the consistent initialization problem is abbreviated as IP; the initial value problem is abbreviated as IVP.

2.1 The IP

The IP can be defined as follows: given information on the initial state $\dot{\mathbf{x}}(t_0)$, $\mathbf{x}(t_0)$, $\mathbf{y}(t_0)$ that is sufficient, in a mathematical sense, to find a unique solution (at least locally) to Equation (1) on an interval starting at $t = t_0$, determine the complete initial state $\dot{\mathbf{x}}(t_0)$, $\mathbf{x}(t_0)$, $\mathbf{y}(t_0)$ corresponding to this unique solution [Brenan et al. 1996]. The values $\dot{\mathbf{x}}(t_0)$, $\mathbf{x}(t_0)$, $\mathbf{y}(t_0)$ are referred to as the consistent initial conditions.

The initial conditions $\dot{\mathbf{x}}(t_0), \mathbf{x}(t_0), \mathbf{y}(t_0)$ must satisfy

$$\mathbf{f}(\dot{\mathbf{x}}(t_0), \mathbf{x}(t_0), \mathbf{y}(t_0), t_0) = \mathbf{0}. \tag{2}$$

This is, however, not always enough for consistency of the initial conditions. For certain systems of equations, differentiating a subset of the equations defined by (1) leads to additional equations, also called the hidden constraints, that must also be satisfied by the initial conditions [Pantelides 1988].

The number of (dynamic) degrees of freedom r of a set of $n + m$ equations (1), is the number of initial conditions from the $2n + m$ initial conditions $\dot{\mathbf{x}}(t_0)$, $\mathbf{x}(t_0)$, $\mathbf{y}(t_0)$ (or: $\dot{x}_0(t_0), \ldots, \dot{x}_{n-1}(t_0), x_0(t_0), \ldots, x_{n-1}(t_0), y_0(t_0), \ldots, y_{m-1}(t_0)$) that can be arbitrarily specified and still allow consistent initialization [Unger et al. 1995]. When there are no hidden constraints, the number of dynamic degrees of freedom is n. Hidden constraints lead to $r < n$.

2.2 The IVP

Given the equation

$$\mathbf{f}(\dot{\mathbf{x}}(t), \mathbf{x}(t), \mathbf{y}(t), t) = \mathbf{0}, \tag{3}$$

determine $\mathbf{x}(t), \mathbf{y}(t)$ on an interval starting at $t = t_0$ for given consistent initial conditions $\dot{\mathbf{x}}(t_0)$, $\mathbf{x}(t_0)$, $\mathbf{y}(t_0)$.

2.3 Unknowns in the IP

In this section, the IP is further discussed for equations that do not require additional differentiation in order to allow consistent initial conditions to be determined (see Section 7). The variables that occur in the equations of the IP are divided into two categories: the unknown and the known variables. In the IP, the values of the unknown variables are determined as a function of the values of the known variables. Therefore, the values of the known variables are not affected by solving the IP. A frequently occurring IP is given by considering $\mathbf{x}(t_0)$ as known, and

$\dot{\mathbf{x}}(t_0), \mathbf{y}(t_0)$ as unknown in (2). For ODEs (ordinary differential equations)

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), t), \tag{4}$$

this IP is trivial: the value of $\dot{\mathbf{x}}(t_0)$ can be calculated by assigning $\mathbf{f}(\mathbf{x}(t_0), t_0)$ to $\dot{\mathbf{x}}(t_0)$. For DAEs, numerical solvers may be needed to solve (2) for $\dot{\mathbf{x}}(t_0), \mathbf{y}(t_0)$.

In hybrid systems, the IP may need to be solved in the following situations:

—After the differential variables are initialized for the first time, usually at time zero.

—At discrete events, after changing to a different set of equations and/or unknowns, or after a change in the values of one or more of the known variables in the equations.

The fact that the value of $\mathbf{x}(t_0)$ remains unchanged when the IP is solved for $\dot{\mathbf{x}}(t_0)$ and $\mathbf{y}(t_0)$, is sometimes referred to as the 'continuity assumption' for $\mathbf{x}$.

Changes in the equations can require changes in the knowns and unknowns for the IP. Consider the single equation

$$\dot{x} = 1 \tag{5}$$

with one degree of freedom, and x as known variable and $\dot{x}$ as unknown. If this equation is changed at a discrete event into

$$x = 1, \tag{6}$$

the equation can only be solved for x. Therefore, x becomes unknown, and there are no longer any known variables; the number of degrees of freedom is reduced from one to zero.

In Equations (5) and (6), there is only one way to define a solvable IP. For equation

$$\dot{x} = -x + 1$$

however, x may be considered known or unknown. The latter case could occur when steady-state conditions are determined. For steady-state conditions, an additional equation defines the value of $\dot{x}$ to be 0:

$$\begin{aligned} \dot{x} &= -x + 1 \\ \dot{x} &= 0. \end{aligned}$$

In the resulting new IP, x changes from a known to an unknown variable. Its value is then determined by the equations (and equals 1). The steady-state IP is specified in a general way by

$$\mathbf{f}(\dot{\mathbf{x}}(t_0), \mathbf{x}(t_0), \mathbf{y}(t_0), t_0) = \mathbf{0} \tag{7a}$$

$$\dot{\mathbf{x}}(t_0) = \mathbf{0}, \tag{7b}$$

with unknowns $\dot{\mathbf{x}}(t_0)$, $\mathbf{x}(t_0)$, and $\mathbf{y}(t_0)$. The number of dynamic degrees of freedom is zero.

2.4 Unknowns in the IVP

The variables that occur in the equations of the IVP are divided into two categories: the unknown and the known variables. In the IVP, the values of the unknown variables are determined as a function of the consistent initial conditions, the time, and the known variables; the values of the known variables are not affected by solving the IVP. The set of unknown variables for the IVP is usually different from the set of unknown variables for the IP for two reasons: first, differential variables (occurring in the active equations) are always unknown in the IVP, but they can be known in the IP; second, derivatives (occurring in the active equations) are considered as unknown variables for the IP, whereas for the IVP, they are not separate variables, and therefore they are not in the set of unknowns for the IVP.

2.5 Example

Consider a tank containing a volume V of liquid, an inlet feeding a flow of 2, and an outlet with a flow Q. The model of the tank system is:

$$\begin{aligned} \dot{V}(t) &= 2 - Q(t) \\ Q(t) &= \sqrt{V(t)}. \end{aligned}$$

If at time 0, the volume of the tank $V(0)$ is considered as known, the IP is specified by:

$$\begin{aligned} \dot{V}(0) &= 2 - Q(0) \\ Q(0) &= \sqrt{V(0)}, \end{aligned}$$

with $\dot{V}(0)$ and $Q(0)$ as unknowns. When $V(0) = 9$, then $Q(0) = 3$ and $\dot{V}(0) = -1$.

In the case of steady-state initialization, the IP becomes:

$$\begin{aligned} \dot{V}(0) &= 2 - Q(0) \\ Q(0) &= \sqrt{V(0)} \\ \dot{V}(0) &= 0, \end{aligned}$$

where $\dot{V}(0)$, $V(0)$, and $Q(0)$ are the unknowns. This leads to the consistent initial conditions $\dot{V}(0) = 0$, $Q(0) = 2$, and $V(0) = 4$.

For the IVP of the same system, the unknowns are $V(t)$ and $Q(t)$:

$$\begin{aligned} \dot{V}(t) &= 2 - Q(t) \\ Q(t) &= \sqrt{V(t)} \end{aligned}$$

where $\dot{V}$ denotes the derivative function of V.

3. UNKNOWNS IN HYBRID LANGUAGES

An introduction to the functionality of different hybrid languages can be found in [Mosterman 1999; Gueguen and Lefebvre 2001; van Beek and Rooda 2000]. The distinction between discrete and continuous variables, that is usually present in hybrid languages, may be made in the type of the variables [IEEE 1999], but also in the place where they are used (e.g. Tasks versus Modules in gPROMS [Barton and Pantelides 1994]). In hybrid automata [Alur et al. 1995], discrete variables may be referred to as 'locations'. In hybrid languages, continuous variables are unknown

in the IVP; discrete variables are known in the IVP and usually also known in the IP. There are, however, languages, such as Modelica [Mattsson et al. 1998], that use equations to determine the values of discrete variables.

The majority of hybrid languages is based on the assumption that discontinuities in differential variables at discrete events are modeled by explicit mappings. An example of such languages are formal languages, such as hybrid automata [Alur et al. 1995] or Petri nets [David and Alla 2001]. The equations considered are usually of the form $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, t), \mathbf{y} = \mathbf{g}(\mathbf{x}, t)$. This implies that the differential variables can be independently initialized. The unknowns and knowns for the IP are fixed with $\mathbf{x}$ considered known, and $\dot{\mathbf{x}}, \mathbf{y}$ considered unknown. When there are algebraic equations restricting the allowed new values of the differential variables (see Section 1), explicit remapping of differential variables forces the modeler to solve the algebraic equations. To overcome this difficulty, many different language elements are used in hybrid languages to deal with the different types of algebraic restrictions on differential variables, see for example [Barton and Pantelides 1994; IEEE 1999; Mattsson et al. 1998]. There are languages that have language elements to denote the value of a differential variable before and/or after a discrete event, such as old*(var)*/new*(var)*, or var^-/var^+, where *var* denotes a (differential) variable. These expressions can then be used in instantaneous equations that specify the relations between the values of differential variables before and after a discrete event.

Another type of dependency between differential variables occurs at steady-state initialization. In steady-state initialization, the differential variables, which are usually known for the IP, temporarily become unknown. Some simulation languages (e.g. [IEEE 1999]) start by default with the steady-state IP as specified in Equation (7), with $\dot{\mathbf{x}}(t_0)$, $\mathbf{x}(t_0)$, $\mathbf{y}(t_0)$ unknown. In other simulation languages, the unknowns $\dot{\mathbf{x}}(t_0)$, $\mathbf{y}(t_0)$ in the first IP, when the system is first initialized, are no different from the unknowns in subsequent IPs. Usually, additional language elements are available in simulation languages to change from steady-state initialization to non-steady-state initialization. The instantaneous steady state equations ($\dot{\mathbf{x}} = 0$) are usually not explicitly modeled. Instead, special purpose language elements are used. Also additional language elements may be available to allow the temporary addition of equations to the IP, and to specify which differential variables become unknown as a result. In the case that equations are removed from the system, the associated continuous variables may be specified as undefined [Barton 1992].

4. A LANGUAGE ELEMENT FOR THE DECLARATION OF UNKNOWNS

The different types of dependencies between differential variables can be modeled by means of a single language element for the declaration of unknowns. Using unknown declarations, it is no longer necessary to introduce a static separation between discrete and continuous variables. Unknown declarations consist of a list of unknowns between bars, which can be declared wherever equations can be declared. An unknown is either a variable or a derivative of type real, or based on type real (tuples, records, arrays of reals), optionally postfixed with a prime character to denote a derivative. Not all unknowns need to be explicitly declared as unknown in an unknown declaration.

For the IVP, the differential variables — that is the variables that occur differentiated in the active equations (see Section 5.3) — are by definition unknown. The reason for this is that in differential (algebraic) equations, the differential variables always need to be solved as a function of time. The derivatives that occur in the active equations are not considered as separate variables. Instead, the dot operator in $\dot{\mathbf{x}}$ is considered to transform the function $\mathbf{x}$ into its derivative function—so that in $\dot{\mathbf{x}}$, the unknown is $\mathbf{x}$. Therefore, declarations of derivatives as unknown are disregarded for the IVP.

For the IP, the derivatives that occur in the active equations are by definition unknown. There are two reasons for this. First, a derivative that would be allowed to be known would behave as a variable and could be assigned a value. This is inconsistent with the fact that derivatives are not considered as separate variables in the IVP. Second, modeling becomes easier when the modeler needs not explicitly declare all derivatives as unknown. We have not found any realistic models that would benefit from allowing derivatives to be known.

4.1 Informal semantics

A run of a hybrid model is an alternating sequence of discrete and continuous phases. In both phases, there is a set of unknowns. The variables and derivatives that are in the set of unknowns are considered unknown for the IP or IVP; all other variables are considered known. The value of a known variable is determined by a mapping. The value of an unknown is determined by an IP or IVP.

In a discrete phase, model time is constant and statements are executed (or actions/events take place). The active equations and the set of unknowns define an IP that defines the values of the unknowns as a function of the knowns. When no further statements can be executed without advancing model time, the discrete phase terminates and a continuous phase starts. In a continuous phase, the active equations and the set of unknowns define an IVP, which defines the values of the unknowns as a function of the consistent initial conditions, the time, and the knowns. When a time-event or state-event occurs, the continuous phase terminates, and a discrete phase starts.

The set of unknowns for the IP contains the derivatives that occur in the active equations, together with the variables that are explicitly declared as unknown in the active unknown declarations (see Section 5.3). The value of an unknown needs to be defined only when the value is actually required in an expression of an executing statement.

The set of unknowns for the IVP contains the variables that occur differentiated in the active equations (the differential variables), together with the variables that are explicitly declared as unknown in the active unknown declarations. Only algebraic variables need to be explicitly declared as unknown for the IVP, since the differential variables are by definition in the set of unknowns.

When a variable changes from unknown to known, its value changes from being determined by an IP or IVP, to being determined by a mapping. The value defined by the mapping is the value of the variable that was defined by the IP or IVP, just before the variable switched from unknown to known. Such a switch usually takes place for differential variables, when a continuous phase switches to a discrete phase, which implies a switch from an IVP to an IP. The differential variables then

usually switch from unknown to known, because they are by definition unknown in IVPs, whereas they are usually known in IPs. The removal of temporary additional equations, that determine the values of differential variables, can also lead to such a phenomenon. Examples are bodies colliding at a single point of time and steady-state initialization (see sections 6.1 and 6.2).

The switch from a discrete phase to a continuous phase cannot cause variables to switch from unknown to known. However, the consistent initial conditions of the IVP ($\dot{\mathbf{x}}(t_0)$, $\mathbf{x}(t_0)$, $\mathbf{y}(t_0)$ in Section 2.2) are defined to be the values that are determined by the preceding IP at the time point of the switch to the IVP. Therefore, the *starting* values of the unknowns in the IVP are in fact considered known.

Declaring a derivative as unknown in an unknown declaration has no effect on the set of unknowns for the IP or IVP, and therefore does not change the semantics of the model. Derivatives occurring in active equations may, however, be explicitly declared as unknown to clarify the structure of the system of equations. This may help the modeler to detect structurally singular systems of equations (see Section 7).

5. THE χ LANGUAGE

Unknown declarations will be implemented in the newest version of the χ language. An older version of the language is discussed in [van Beek and Rooda 2000]. The χ language has been designed from the start as a hybrid language that can be used for specification, verification [Bos and Kleijn 2000a; 2002], simulation, and real-time control [Hofkamp 2001] of discrete-event systems [van de Mortel-Fronczak et al. 1995], continuous-time systems, and combined discrete-event / continuous-time systems [van Beek and Rooda 2000]. The language is based on mathematical concepts with well defined semantics [Bos and Kleijn 2000b; 2002]. The discrete-event part of χ is based on Communicating Sequential Processes [Hoare 1978], the continuous-time part on differential algebraic equations. Processes are parameterized and can be grouped into systems; channels and shared variables are used for inter-process communication and synchronization. High level data types are available such as arrays, lists and sets along with many associated operators.

The χ simulator is described in [Fábián 1999] and [Naumoski and Alberts 1998]. It has been successfully applied to a large number of industrial cases, such as an integrated circuit manufacturing plant [Rulkens et al. 1998], a brewery [van Beek et al. 2002], and a fruit juice blending and packaging plant [Fey 2000]. In the sequel, only a minimal subset of the χ language is used. The syntax and operational semantics of the language constructs are explained in an informal way. The models are kept as short as possible, showing only the essentials.

5.1 Process and system syntax

A process may consist of a variable declaration part, a continuous-time part (differential algebraic equations), a discrete-event part (statements), or any combination of the different parts. The syntax of a process consisting of all parts is:

```
proc name(parameter declarations) =
|[ variable declarations
 | equations
```

```
 |  statements
]|
```

A system may consist of a variable and channel declaration part, and a process instantiation part. Process instantiations are separated by the parallel symbol '||':

```
syst name(parameter declarations) =
|[  variable and channel declarations
 |  process instantiations
]|
```

5.2 Equation syntax

A somewhat simplified syntax of equations in χ follows below. The syntax is described in extended BNF where $\{X\}$ denotes zero or more repetitions of construct X, and $[X]$ denotes zero or one occurrence of construct X. Literal characters are enclosed in quotes.

```
derivative  ::= variable primechar
unknown     ::= variable | derivative
unknownDecl ::= '|' unknown {, unknown} '|'
guardedEq   ::= '[' b '->' UEq ']'
UEq         ::= unknownDecl
              | guardedEq
              | re '=' re
              | UEq, UEq
```

where `b` denotes a boolean expression; `re` denotes an expression of type `real` (or based on `real`), `primechar` denotes the prime character, `guardedEq` denotes a guarded equation, and `UEq` denotes unknown declarations and/or equations. A guarded (or in other words conditional) equation consists of one or more equations or unknown declarations that are prefixed by a boolean expression, called a guard, as in `[b -> x = 1]`. The arrow '`->`' separates the guard from the equation(s). A guard is open when its value evaluates to true and is closed otherwise.

5.3 Active equations and active unknown declarations

When the equations of a model are solved, only the *active* equations and the *active* unknown declarations are taken into account. The active equations (and unknown declarations) are the unguarded equations (and unguarded unknown declarations) together with those guarded equations (and unknown declarations) that have an open guard (value of guard is true).

6. EXAMPLES

The semantics is illustrated by means of the following example. The equation part is separated from the declaration part by the symbol `|`, and comments are prefixed by `//`:

```
proc P =
|[ V: real:= 0.0
 , x: real:= 0.0
 , Q: real
```

```
 , b: bool:= true
 | |Q|
 , V' = 2 - Q
 , Q  = sqrt (V)
 , [ b     ->      x' = 1 - x ]
 , [ not b -> |x|, x  = 2 ]
]|
```

Variables V, x, and Q are declared as reals. Function `sqrt` represents the square root function. Variable b is a boolean variable, that is initialized to true. When b is true, the set of active equations consist of equations $V' = 2-Q$, $Q = \sqrt{V}$, $x' = 1-x$. The unknowns for the IP are then V', x' (by definition), and Q (explicitly declared as unknown). This means that the values of V and x remain unchanged when the consistent initial conditions are determined by solving the IP. The unknowns for the IVP are V, x (by definition), and Q (explicitly declared as unknown).

When boolean variable b is false, the set of active equations consist of Equations $V' = 2-Q$, $Q = \sqrt{V}$, $x = 2$. Variable x is now an algebraic variable. The unknowns for the IP are V', Q, and x. The unknowns for the IVP are V, Q, and x.

6.1 Modeling colliding bodies

Consider a body of mass m_0 and velocity v_0, with applied force 1, moving towards another body of mass m_1 and velocity v_1, and colliding with that body. The positions of the bodies are denoted by x_0 and x_1, respectively. Variables $v_{\text{old}0}$ and $v_{\text{old}1}$ denote the velocities of the two bodies just before the collision. The model follows below.

```
type pos = real    // position
,    vel = real    // velocity
,   mass = real

proc C0(m0,m1: mass, e: real) =
|[ x0: pos := 0.0
 , x1: pos := 1.0
 , v0: vel := 0.0
 , v1: vel := 0.0
 , vold0,vold1: vel
 | x0' = v0, v0' = 1.0
 , x1' = v1, v1' = 0.0
 | *[ true
      -> nabla x0 >= x1; vold0:= v0; vold1:= v1
       ; { |v0,v1|
         , v0 - v1 = -e * (vold0 - vold1)
         , m0*v0 + m1*v1 = m0*vold0  + m1*vold1
         }
       ; nabla x0 < x1
   ]
]|
```

In the declaration part of the specification, the positions of the bodies are initialized to 0 and 1, respectively; the velocities are initialized to 0. The meaning of `nabla` $x_0 \geq x_1$, as the first statement of the infinite repetition `*[true -> ...]`, is

that the process waits until the value of expression $x_0 \geq x_1$ becomes true. This is the case when the collision occurs. Subsequently, $v_{\text{old}0}$ and $v_{\text{old}1}$ are set to v_0 and v_1, respectively. The following statement to be executed is an instantaneous set of unknown declarations and equations $\{|v_0, v_1|, \cdots\}$. Unknown declaration $|v_0, v_1|$ defines variables v_0 and v_1 as unknown; the first equation, $v_0 - v_1 = -e(v_{\text{old}0} - v_{\text{old}1})$, relates the differences in the velocities before and after the collision as a function of the restitution coefficient $e \in [0, 1]$; the second equation specifies the conservation of momentum. The semantics is that the unknowns v_0 and v_1 are added to the set of unknowns, the two instantaneous equations are added to the active equations, and the IP is solved for the unknowns. The two instantaneous equations define v_0 and v_1 as functions of $v_{\text{old}0}$, $v_{\text{old}1}$, m_0, m_1, and e. After execution of the instantaneous equation statement, variables v_0 and v_1 (and x_0', x_1', since $x_0' = v_0$, $x_1' = v_1$) have the values that reflects the state directly after the collision; v_0 and v_1 change from unknown into known variables again (for the IP), and the two additional equations are no longer in the set of active equations. A functionally equivalent specification of the colliding objects in which no instantaneous equations are used is:

```
proc C1(m0,m1: mass, e: real) =
|[ x0: pos = 0.0
 , x1: pos = 1.0
 , v0: vel = 0.0
 , v1: vel = 0.0
 , vold0,vold1: vel
 | x0' = v0, v0' = 1.0
 , x1' = v1, v1' = 0.0
 | *[ true
      -> nabla x0 >= x1; vold0:= v0; vold1:= v1
       ; v0 := (m0*vold0 - e*m1*vold0
               +m1*vold1 + e*m1*vold1)/(m0 + m1)
       ; v1 := (m0*vold0 + e*m0*vold0
               -e*m0*vold1 + m1*vold1)/(m0 + m1)
       ; nabla x0 < x1
    ]
]|
```

A first disadvantage of this model is that the modeler is forced to calculate the new values for v_0 and v_1 directly after the collision as explicit functions of $v_{\text{old}0}$, $v_{\text{old}1}$, m_0, m_1, and e. A second disadvantage is that the model no longer reflects the physical laws that are valid at the point of collision.

6.2 Steady-state initialization

Consider a slightly modified version of the tank system from Section 2.5. Process T has an incoming flow Q_i and an outgoing flow Q_o:

```
proc T( ref steady: bool
      , ref Qi,Qo: real
      ) =
|[ V: real
 | [ steady -> |V|, V' = 0 ]
 , |Qo|
```

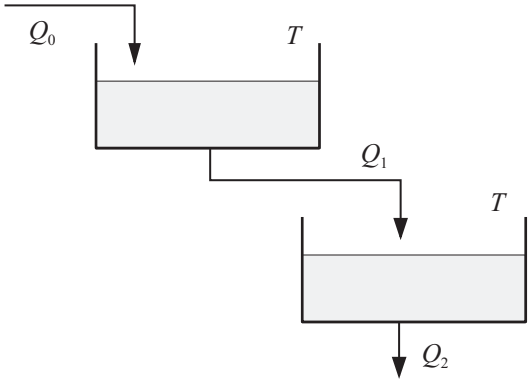

Fig. 1. A two tank system.

```
  , V' = Qi - Qo,
  , Qo = sqrt(V)
 ]|
```

Boolean variable *steady* is true when steady-state initialization is used; it is false when the value of differential variable V should be unaffected by solving the IP. The keyword `ref` can be prefixed to a process parameter. The meaning is that such a parameter is a shared variable, that can be used in more than one process.

If boolean variable *steady* is true, the active equations and active unknown declarations are equivalent to:

$$|V, Q_o|,\ V' = 0,\ V' = Q_i - Q_o,\ Q_o = \sqrt{V}$$

with unknowns for the IP: V' (by definition), V, and Q_o (declared unknown explicitly).

If boolean variable *steady* is false, the active equations and unknown declarations are equivalent to:

$$|Q_o|,\ V' = Q_i - Q_o,\ Q_o = \sqrt{V}$$

with V' and Q_o unknown for the IP, and V known for the IP, so that the value of differential variable V remains unchanged when the IP is solved.

Figure 1 shows two tanks that are connected in such a way that the outgoing flow of the first tank is connected to the incoming flow of the second tank. System *TwoTanks*, specified below, is a model of the two connected tanks. Two tank processes are instantiated in such a way that the outgoing flow of the first tank process T is connected to the incoming flow of the second process T. Variables Q_0, Q_1, Q_2 are defined as variables in the system. They are shared between the two tank process instantiations $T(\cdots)$. Process *Source* defines the value of Q_0 to be 2. Local variable *steady* is declared as being initially true, so that the system is initialized in steady state. Variable *steady* is shared with process *Ini*. In this process, the value of the variable is set to false, so that the additional steady state equations $|V|,\ V' = 0$ are made inactive. First, the initialization to true at the declaration of variable *steady* in system *TwoTanks* is executed. As a result, the

values of Q_0, Q_1, Q_2, V_0, V_1 will be 2, 2, 2, 4, 4, respectively. After that, the assignment of false to variable *steady* in process *Ini* is executed.

```
proc Source(ref Q: real) =
|[ |Q|, Q = 2 ]|

proc Ini(ref steady: bool) =
|[ steady:= false ]|

syst TwoTanks() =
|[ steady: bool:= true
 , Q0,Q1,Q2: real
 | Source(Q0) || Ini(steady)
|| T(steady, Q0, Q1)
|| T(steady, Q1, Q2)
]|
```

7. HIGHER INDEX SYSTEMS

Most hybrid simulation languages cannot deal with higher index systems. Unfortunately, many models of physical systems are of a higher index. In this section, first the higher index problem is explained; it is related to dependencies between differential variables. Second, it is explained that unknown declarations can help to deal with higher index systems in two ways:

(1) Unknown declarations enable the modeler to partition the differential variables into dependent and independent variables by choosing the dependent differential variables and declaring them as unknown. The remaining (independent) differential variables can then be freely initialized by the modeler. The simulator can calculate the values of the dependent variables.

(2) Unknown declarations can be used to clarify the structure of a system of equations. This can be done by prefixing unknown declarations to equations. In this way, the modeler can detect higher index systems. Together with systematic modeling techniques, this may enable the modeler to keep the index of the model equations low.

In the previous sections, the assumption was made that initial conditions $\dot{\mathbf{x}}(t_0)$, $\mathbf{x}(t_0)$, $\mathbf{y}(t_0)$ are consistent if they satisfy Equation (2). For higher index systems, initial conditions need also satisfy the hidden constraints, that are obtained after differentiation of Equation (1). The need to differentiate (a subset of) the system of equations in order to determine the consistent initial conditions is related to the index of the system of equations. In general, the higher the index, the greater the numerical difficulty that is encountered when trying to solve the system numerically. Several index definitions exist. In most cases, the differential index [Gear 1988] is used. This index is rather straightforward to determine, due to its constructive

definition. Consider the system of equations

$$\mathbf{f}(\dot{\mathbf{x}}, \mathbf{x}, \mathbf{y}, t) = \mathbf{0} \tag{8a}$$
$$\frac{d}{dt}\mathbf{f}(\dot{\mathbf{x}}, \mathbf{x}, \mathbf{y}, t) = \mathbf{0} \tag{8b}$$
$$\vdots$$
$$\frac{d^j}{dt^j}\mathbf{f}(\dot{\mathbf{x}}, \mathbf{x}, \mathbf{y}, t) = \mathbf{0}, \tag{8c}$$

where Equation (8a) is as defined in the beginning of Section 2. This system can be written as

$$\mathbf{f}_0(\dot{\mathbf{x}}, \mathbf{x}, \mathbf{y}, t) = \mathbf{0} \tag{9a}$$
$$\mathbf{f}_1(\ddot{\mathbf{x}}, \dot{\mathbf{y}}, \dot{\mathbf{x}}, \mathbf{x}, \mathbf{y}, t) = \mathbf{0} \tag{9b}$$
$$\vdots$$
$$\mathbf{f}_j(\mathbf{x}^{(j+1)}, \mathbf{y}^{(j)}, \ldots \ddot{\mathbf{x}}, \dot{\mathbf{y}}, \dot{\mathbf{x}}, \mathbf{x}, \mathbf{y}, t) = \mathbf{0}. \tag{9c}$$

For the determination of the differential index of (8a), the variables $\mathbf{x}^{(j+1)}$, $\mathbf{y}^{(j)}$, $\ldots \ddot{\mathbf{x}}$, $\dot{\mathbf{y}}$, $\dot{\mathbf{x}}$ are treated as purely algebraic variables, depending on $\mathbf{x}$, $\mathbf{y}$, and t. The differential index is then defined as the smallest value of j for which system (9) uniquely defines $[\dot{\mathbf{x}}\ \dot{\mathbf{y}}]^{\mathrm{T}}$ as a function of $\mathbf{x}$, $\mathbf{y}$, and t.

Systems of equations that have an index greater than one are called *higher index* systems. A common feature of higher index systems is the presence of hidden constraints, that further restrict the initial conditions that satisfy (8a). The hidden constraints can be obtained by differentiation of (a subset of) the equations and subsequent algebraic manipulations [Gear 1988; Pantelides 1988]. Consider

$$\dot{x} = y \tag{10a}$$
$$x = 1. \tag{10b}$$

The differential index is 2. Differentiation of $x = 1$ leads to $\dot{x} = 0$, which leads to $y = 0$. A second differentiation leads to $\dot{y} = 0$, so that after two differentiations the ODE $\dot{x} = 0$, $\dot{y} = 0$ is obtained. Initial conditions that satisfy Equations (10) are $\dot{x}(t_0) = y(t_0) = x(t_0) = 1$. These initial conditions, however, do not satisfy the hidden constraint $\dot{x} = 0$, which is obtained after differentiation. Therefore, consistent initial conditions need to satisfy

$$\begin{aligned} \dot{x} &= y \\ x &= 1 \\ \dot{x} &= 0, \end{aligned}$$

and are $\dot{x}(t_0) = y(t_0) = 0$, and $x(t_0) = 1$. Even DAEs of (differential) index one may require differentiation in order to reveal hidden constraints, and to allow consistent initialization. Consider

$$\dot{x}_0 = \dot{x}_1 \tag{11a}$$
$$x_0 = 1. \tag{11b}$$

Initial conditions that satisfy Equations (11) are $\dot{x}_0(t_0) = \dot{x}_1(t_0) = x_0(t_0) = 1$, and $x_1(t_0) = 0$. These initial conditions, however, do not satisfy the hidden constraint $\dot{x}_0 = 0$, which is obtained after differentiation. Consistent initial conditions need to satisfy

$$\begin{aligned} \dot{x}_0 &= \dot{x}_1 \\ x_0 &= 1 \\ \dot{x}_0 &= 0, \end{aligned}$$

and are $\dot{x}_0(t_0) = \dot{x}_1(t_0) = 0$, $x_0(t_0) = 1$, and $x_1(t_0) = 0$. The system has one dynamic degree of freedom: $x_1(t_0)$ can be given an arbitrary initial value. The differential index, however, is one, since differentiation of $x_0 = 1$ leads to the ODE $\dot{x}_0 = 0$, $\dot{x}_1 = 0$.

For the determination of consistent initial conditions it is more relevant to consider the smallest value of j for which system (9) uniquely (at least locally) defines $[\dot{\mathbf{x}}\ \mathbf{y}]^{\mathrm{T}}$ as a function of $\mathbf{x}$, and t. This value of j is the number of times that the original DAE (8a) needs to be differentiated in order to be able to determine the consistent initial conditions. If a DAE

$$\mathbf{f}(\dot{\mathbf{x}}, \mathbf{x}, \mathbf{y}, t) = \mathbf{0}$$

uniquely (at least locally) defines $[\dot{\mathbf{x}}\ \mathbf{y}]^{\mathrm{T}}$as a function of $\mathbf{x}$ and t, the DAE does not contain hidden constraints. This is the case if and only if $\partial\mathbf{f}/\partial\mathbf{z}$ is nonsingular, where $\mathbf{z} = [\dot{\mathbf{x}}\ \mathbf{y}]^{\mathrm{T}}$. The initial value $\mathbf{x}(t_0)$ of such a DAE can be arbitrarily chosen; consistent initial conditions $\dot{\mathbf{x}}(t_0), \mathbf{x}(t_0), \mathbf{y}(t_0)$ need only satisfy the original equations. This can be seen as follows: differentiation of the function that uniquely defines $[\dot{\mathbf{x}}\ \mathbf{y}]^{\mathrm{T}}$ as a function of $\mathbf{x}$ and t, yields a function that defines $[\ddot{\mathbf{x}}\ \dot{\mathbf{y}}]^{\mathrm{T}}$ as a function of $\dot{\mathbf{x}}$, $\mathbf{x}$, and t. Therefore, for every set of initial conditions $\dot{\mathbf{x}}(t_0), \mathbf{x}(t_0), \mathbf{y}(t_0), t_0$ that satisfy the original equation, values of $\ddot{\mathbf{x}}(t_0)$ and $\dot{\mathbf{y}}(t_0)$ can be chosen that satisfy the additional equations.

7.1 Structural analysis

Another way of analyzing the properties of systems of equations is by means of *structural* analysis. Structural analysis distinguishes only zero and nonzero values [Unger et al. 1995]; only the presence of variables in equations is taken into account, not the numerical value of the corresponding coefficients in the Jacobian matrix. Structural analysis of equations is, in general, a more efficient way to detect higher index systems than numerical analysis of the Jacobian matrix. The Pantelides algorithm [Pantelides 1988], that is based on structural analysis, identifies the minimal subset of equations that must be differentiated for the determination of consistent initial conditions. Pantelides shows that DAEs that are structurally singular with respect to $[\dot{\mathbf{x}}\,\mathbf{y}]^{\mathrm{T}}$ require differentiation of the equations in order to reveal the hidden constraints. A system of equations is defined as structurally singular with respect to a certain set of variables if it contains a subset of equations that is structurally singular with respect to the same set of variables. A subset of equations is called structurally singular with respect to a set of variables if the number of these variables that occur in the equation subset is smaller than the number of equations in the subset. The Pantelides algorithm tries to assign each of the variables of $[\dot{\mathbf{x}}\ \mathbf{y}]^{\mathrm{T}}$

to a different equation. If this is possible, the system of equations is structurally regular. If this is not possible, the algorithm finds the equations that need to be differentiated in order to enable consistent initialization. Structural regularity of equations with respect to $[\dot{\mathbf{x}}\ \mathbf{y}]^{\mathrm{T}}$ is a necessary, but not sufficient, requirement to allow consistent initialization without differentiation. The reason for this is that a system that is structurally regular may still have a singular Jacobian matrix. Only few simulation languages actually provide structural analysis and automatic index reduction by means of differentiation in order to reduce the index and allow consistent initialization. The Dymola [Elmqvist et al. 2000] simulation language is one of them. ABACUSS I [Feehery and Barton 1996] did provide this feature, using the dummy derivative method [Mattsson and Söderlind 1993], but the feature is not available in its successor ABACUSS II [Clabaugh 2001]. For the many simulation languages that do not provide automatic index reduction, the modeler needs to do structural analysis and index reduction him/herself, and/or use systematic modeling techniques that keep the index low. Such a technique is described in [Moe 1995]. A key element of this technique is the assignment of the variables of $[\dot{\mathbf{x}}\ \mathbf{y}]^{\mathrm{T}}$ to the equations. The chosen assignment of variables to equations is essentially a bookkeeping method; it is not part of the model. The chosen assignments could be informally indicated by means of comments to the model.

7.2 Declaration of unknowns

Using the proposed language element for the declaration of unknowns, the chosen assignment of variables to equations can be made part of the model. For this purpose, consider that the variables represented by $\mathbf{y}$ in Equation (1) are algebraic variables, which do not occur differentiated in the equations. Therefore, they need to be declared as unknown. The variables represented by $\dot{\mathbf{x}}$ can be explicitly declared as unknown without changing the semantics of the model (see Section 4.1). An unknown declaration can be put on the same line as an equation in which the unknown occurs. This is interpreted as 'assigning' the unknown to the equation. The exact location where variables are declared as unknown has no effect on the semantics of the model, unless they are declared as part of a guarded equation. In such a case, it depends on the state of the guards whether or not the unknown declaration is active. By using unknown declarations to assign the derivatives represented by $\dot{\mathbf{x}}$ and the algebraic variables represented by $\mathbf{y}$ to equations in which they occur, the assignment of these variables becomes part of the model. This has the advantage that the structure of the model is clarified, systematic modeling techniques are facilitated, and structural singularities—which lead to higher index systems—can be made visible. Even in the case of automatic index reduction by simulation languages, the modeler needs to be able to indicate which of the differential variables are explicitly given independent initial values, and which of the differential variables are considered dependent. This is necessary because the differential variables can no longer be initialized independently in higher index systems. The differential variables that are considered dependent can be indicated by declaring them unknown in an unknown declaration.

7.3 PD controller example

The examples show how unknown declarations can be used to assign each derivative and algebraic variable to a unique equation. If this is not possible, structural singularities can be identified. As an example of a higher index system, consider the following PD (proportional differential) controller. A horizontal force F is applied to a body of mass 1 on a flat surface, without friction. The position and velocity of the body are denoted by x, and v, respectively. The control objective is to keep the body at a given position x_{set}. The control error and the control output are denoted by e and u, respectively. The model is:

```
type pos   = real
,    vel   = real
,    force = real

proc PID0( kP,kD: real, xset: pos
         , Fin: real -> force
         )=
|[ x: pos:= 0.0
 , e: pos:= x - xset
 , v: vel:= 0.0
 , F: force, u: real
 | |u,F|,
 , x' = v
 , v' = F - u
 ,  e = x - xset
 ,  u = kP * e + kD * e'
 ,  F = Fin(time)
]|
```

where `time` denotes the current time of the model, k_{P}, k_{D} are parameters, and parameter F_{in} denotes a function from a real (the time) to a force. The derivatives (x', v', e') are unknown for the IP by definition. The algebraic variables u and F are explicitly declared as unknown at the start of the equation set. When initializing e, a problem appears: e is a differential variable, but its value cannot be initialized independently from x. By explicitly declaring the derivatives and algebraic variables as unknown and prefixing them to the equations in which they occur, the structural singularity becomes apparent: the derivatives and algebraic variables cannot each be assigned to a unique equation; the system is an index 2 system, and consistent initialization is only possible after differentiation of equation $e = x - x_{\mathrm{set}}$:

```
proc PID1( kP,kD: real, xset: pos
         , Fin: real -> force
         )=
|[ x: pos:= 0.0, e: pos:= x - xset
 , v: vel:= 0.0
 , F: force, u: real
 | |x'|,  x' = v
 , |v'|,  v' = F - u
 ,         e = x - xset  // Struct. singularity
 , |e',u|, u = kP * e + kD * e'
```

```
 , |F|,    F = Fin(time)
]|
```

The index can be reduced from 2 to 1 by differentiation of equation $e = x - x_{\text{set}}$, which leads to $e' = x'$. By substitution of e' by x', the following index 1 system is obtained:

```
proc PID2( kP,kD: real, xset: pos
         , Fin: real -> force
         )=
|[ x: pos:= 0.0, e: pos
 , v: vel:= 0.0
 , F: force, u: real
 | |x'|, x' = v
 , |v'|, v' = F - u
 , |e|,   e = x - xset
 , |u|,   u = kP * e + kD * x'
 , |F|,   F = Fin(time)
]|
```

Variable e is now an algebraic variable. The equation for u, however, no longer looks like the control law of a PD controller, since x' is used instead of e'. By using a substitute equation [Fábián et al. 2001] for e', the original equation for u needs not be changed. The meaning of substitute equation $e' \leftarrow x'$ is that every time the value of the variable on the left hand side (e') is used in the model, the value of the right hand side expression (x') is used instead. For the simulator, the model thus appears the same as the previous model, in which the modeler performed the substitution of e' by x'; variable e behaves as an algebraic variable. The advantage of using a substitute equation is that the equations of the original model remain unchanged. The resulting model is an index 1 model:

```
proc PID3( kP,kD: real, xset: pos
         , Fin: real -> force
         )=
|[ x: pos:= 0.0, e: pos
 , v: vel:= 0.0
 , F: force, u: real
 | |x'|, x' = v
 , |v'|, v' = F - u
 , |e|,   e = x - xset
 , |u|,   u = kP * e + kD * e'
 , |F|,   F = Fin(time)
 ,      e' <- x'       // substitute e' by x'
]|
```

If the simulator would perform automatic index reduction by means of differentiation, the modeler would still need to indicate which one of the variables e and x becomes unknown, and which one is given an independent initial value. In the model below, e is chosen as the unknown, and x is initialized:

```
proc PID4( kP,kD: real, xset: pos
         , Fin: real -> force
```

```
        )=
|[ x: pos:= 0.0, e: pos
 , v: vel:= 0.0
 , F: force, u: real
 | |e|,
 , |x'|,  x' = v
 , |v'|,  v' = F - u
 ,         e = x - xset
 , |u,e'|, u = kP * e + kD * e'
 , |F|,    F = Fin(time)
]|
```

In this model, e is declared unknown at the top of the system of equations; it is not assigned to any equation. This is because the purpose of assigning unknown declarations is to detect structural singularities of a system of equations with respect to the derivatives and algebraic variables x', v', e', u, F. Since e is a differential variable, it should not be assigned to an equation. However, it (or x) must be declared unknown, because one of the two differential variables can be assigned an initial value; the other differential value is then unknown, so that its value is determined by the equations (in this case $e = x - x_{\text{set}}$). The simulator could use the Pantelides algorithm to determine that equation $e = x - x_{\text{set}}$ is the smallest singular subset of equations with respect to the differential and algebraic variables. The simulator should then derive the additional equation (hidden constraint) $e' = x'$ by differentiation of $e = x - x_{\text{set}}$. Using this additional equation, the IP can be solved for the unknowns. The additional equation leads to an overdetermined set of equations for the IVP, for which special solvers can be used.

7.4 Pendulum example

The pendulum is a well known higher index system. A model of the pendulum of fixed length L in Cartesian coordinates is:

```
const g: real = 9.8

proc Pendulum0(L: real) =
|[ x: real
 , y: real := 0.0
 , vx: real
 , vy: real := 0.0
 , T: real
 | |T,x,vx|
 ,  x' = vx
 ,  y' = vy,
 , vx' = T*x
 , vy' = T*y - g
 , x^2 + y^2 = L^2
]|
```

where g is the gravity constant and T is the tension in the pendulum bar. Although there are 4 differential variables, there are only two dynamic degrees of freedom. This is a result of the algebraic relation $x^2 + y^2 = L^2$ between x and

y. Differentiation of this equation leads to $xx' + yy' = 0$, which can be written as $xv_x + yv_y = 0$. This is again a relation between differential variables that needs to be differentiated to obtain the hidden constraints. In the model, x and v_x are chosen as dependent differential variables that are declared unknown in $|T, x, v_x|$. The other two differential variables y and v_y can be freely initialized. They are both initialized to 0 at their declaration. In order to make clear that the model is a higher index system, the derivatives and algebraic variables x', y', v'_x, v'_y, T can be assigned to the equations. This could, for example, lead to the model:

```
proc Pendulum1(L: real) =
|[ x: real
 , y: real := 0.0
 , vx: real
 , vy: real := 0.0
 , T: real
 | |T,x,vx|
 , |x'|,   x' = vx
 , |y'|,   y' = vy,
 , |vx'|, vx' = T*x
 , |vy'|, vy' = T*y - g
 ,        x^2 + y^2 = L^2
]|
```

Clearly other 'assignments' are also possible. Variable T could be assigned to the third or fourth equation instead of v'_x, or v'_y, respectively. It is clear that none of the derivatives or algebraic variables can be assigned to the last equation. Therefore the model is of a higher index. The models $Pendulum_0$ and $Pendulum_1$ are semantically equivalent.

A simulator could use the Pantelides algorithm [Pantelides 1988] and symbolic differentiation to derive the hidden constraints and calculate the consistent initial conditions for the IP-unknowns $x', y', v'_x, v'_y, T, x, v_x$.

8. CONCLUSIONS

The new language element for the declaration of unknowns is a versatile language element. It makes it easy to specify instantaneous algebraic relations between differential variables. Such relations may be the result of applying time scale or parameter abstractions. Using unknown declarations, the modeler can model the physical laws in their original form as equations, while the simulator can do the solving. Unknown declarations can also be used to model steady state initialization. The use of this single element instead of a number of special purpose language elements makes it easier to develop a simulator, and makes it easier to develop a formal semantics for the purpose of verification. In addition, in the context of higher index systems, unknown declarations are useful for consistent initialization, to clarify the structure of a system of equations, and to help the modeler detect structurally singular systems of equations. Unknown declarations will be implemented in the new release of the hybrid χ simulator.

ACKNOWLEDGMENTS

The authors would like to thank Erjen Lefeber, Ramon Schiffelers and the anonymous referees for their helpful comments on the drafts of the paper.

REFERENCES

Alur, R., Courcoubetis, C., Halbwachs, N., Henzinger, T. A., Ho, P. H., Nicollin, X., Olivero, A., Sifakis, J., and Yovine, S. 1995. The algorithmic analysis of hybrid systems. In *Theor. Comput. Sci.* 138. Springer, 3–34.

Barton, P. I. 1992. The modelling and simulation of combined discrete/continuous processes. Ph.D. thesis, University of London.

Barton, P. I. and Pantelides, C. C. 1994. Modeling of combined discrete/continuous processes. *AIChE 40,* 6, 966–979.

Bos, V. and Kleijn, J. 2000a. Automatic verification of a manufacturing system. *Robotics and Comput. Integr. Manufact. 17,* 3, 185–198.

Bos, V. and Kleijn, J. 2002. Formal specification and analysis of industrial systems. Ph.D. thesis, Eindhoven University of Technology.

Bos, V. and Kleijn, J. J. T. 2000b. Formalisation of a production systems modelling language: the operational semantics of χ core. *Fundamenta Informaticae 41,* 4, 367–392.

Brenan, K. E., Campbell, S. L., and Petzold, L. R. 1996. *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations.* SIAM's Classics in Applied Math. Siam, Philadelphia.

Brown, P. N., Hindmarsh, A. C., and Petzold, L. R. 1998. Consistent initial condition calculation for differential-algebraic systems. *SIAM J. Sci. Comput. 19,* 5, 1495–1512.

Clabaugh, J. 2001. The ABACUSS II syntax manual. Massachussets Institute of Technology. http://yoric.mit.edu/abacuss2/syntax.html.

David, R. and Alla, H. 2001. On hybrid Petri nets. *Discrete Event Dynamic Syst.: Theory & Applicat. 11,* 1-2, 9–40.

Elmqvist, H., Brück, D., and Otter, M. 2000. *Dymola – Dynamic Modeling Language – User's Manual, Version 4.0.* Dynasim AB, Lund, Sweden.

Fábián, G. 1999. A language and simulator for hybrid systems. Ph.D. thesis, Eindhoven University of Technology.

Fábián, G., van Beek, D. A., and Rooda, J. E. 2001. Index reduction and discontinuity handling using substitute equations. *Math. Comput. Modelling Dynamic. Syst. 7,* 2, 173–187.

Feehery, W. F. and Barton, P. I. 1996. A differentiation-based approach to dynamic simulation and optimization with high-index differential-algebraic equations. In *Proc. of 2nd. Int. Workshop on Computational Differentiation.* Santa Fe, 239–252.

Fey, J. J. H. 2000. Design of a fruit juice blending and packaging plant. Ph.D. thesis, Eindhoven University of Technology.

Gear, C. W. 1988. Differential-algebraic equation index transformations. *SIAM. J. Sci. Stat. Comp. 9*, 39–47.

Gueguen, H. and Lefebvre, M. A. 2001. A comparison of mixed specification formalisms. *APII JESA J. Europeen des Systemes Automatises 35,* 4, 381–394.

Hoare, C. A. R. 1978. Communicating sequential processes. *Commun. ACM 21,* 8, 666–677.

Hofkamp, A. T. 2001. Reactive machine control. Ph.D. thesis, Eindhoven University of Technology.

IEEE. 1999. *IEEE Standard VHDL Analog and Mixed-Signal Extensions (IEEE Std 1076.1-1999).* IEEE, New York.

Mattsson, S. E., Elmqvist, H., and Otter, M. 1998. Physical system modeling with Modelica. *Control Eng. Practice 6*, 501–510.

Mattsson, S. E. and Söderlind, G. 1993. Index reduction in differential-algebraic equations using dummy derivatives. *SIAM J. Sci. Comput. 14,* 3, 677–692.

Moe, H. I. 1995. Dynamic process simulation, studies on modeling and index reduction. Ph.D. thesis, University of Trondheim.

Mosterman, P. J. 1999. An overview of hybrid simulation phenomena and their support by simulation packages. In *Hybrid Systems: Computat. Control*. LNCS 1569. Springer, 165–177.

Mosterman, P. J. and Biswas, G. 1997. Principles for modeling, verification, and simulation of hybrid dynamic systems. In *Hybrid Systems: V*. LNCS 1567. Springer, 21–27.

Mosterman, P. J. and Biswas, G. 2002. A hybrid modeling and simulation methodology for dynamic physical systems. *Simulation 78,* 1, 5–17.

Naumoski, G. and Alberts, W. 1998. A discrete-event simulator for systems engineering. Ph.D. thesis, Eindhoven University of Technology.

Pantelides, C. C. 1988. The consistent initialization of differential-algebraic systems. *SIAM J. Sci. Stat. Comput. 9,* 2, 213–231.

Rulkens, H. J. A., van Campen, E. J. J., van Herk, J., and Rooda, J. E. 1998. Batch size optimization of a furnace and pre-clean area by using dynamic simulations. In *SEMI/IEEE Adv. Semiconductor Manufact. Conf.* Boston, 439–444.

Unger, J., Kröner, A., and Marquardt, W. 1995. Structural analysis of differential-algebraic equation systems—theory and applications. *Comput. & Chemical Eng. 19,* 8, 867–882.

van Beek, D. A. and Rooda, J. E. 2000. Languages and applications in hybrid modelling and simulation: Positioning of Chi. *Control Eng. Practice 8,* 1, 81–91.

van Beek, D. A., van den Ham, A., and Rooda, J. E. 2002. Modelling and control of process industry batch production systems. In *15th IFAC Triennial World Cong.* Barcelona. CD-ROM.

van de Mortel-Fronczak, J. M., Rooda, J. E., and van den Nieuwelaar, N. J. M. 1995. Specification of a flexible manufacturing system using concurrent programming. *Concurrent Eng.: Research and Applicat. 3,* 3, 187–194.