Modeling for Optimization of Hybrid Dynamic Networks*

Ashish Agarwal        Ignacio E. Grossmann†

Department of Chemical Engineering, Carnegie Mellon University, Pittsburgh, PA 15213, USA

December 3, 2004

Abstract

Surging interest in hybrid dynamic systems is bringing together a diverse set of solution methods and application domains. The study of combined continuous-discrete systems is complicated by the existence of numerous representational frameworks, each specialized for various application domains and algorithms. For optimization, mixed-integer programming (MIP) is the standard modeling language. Although MIP provides powerful solution methods, its modeling capabilities are severely limited. We provide a richer modeling language, incorporating modular hybrid automata and logic models. These novel constructs are linked to existing solution methods by defining transformations which automatically generate MIPs. Results are provided on a supply network example. The system is represented in the proposed framework and then transformed using the general procedure. Models in the new framework are several times smaller, more intuitive, easier to modify, extensible, and amenable to different algorithms. The net result is an improved modeling language tied to existing optimization algorithms.

Keywords: hybrid automata; modeling languages; optimization; automatic model transformations; supply networks

†Corresponding author. Tel.: +1-412-268-2230; fax: +1-412-268-7139. E-mail addresses: agarwal@cmu.edu (A. Agarwal), grossmann@cmu.edu (I.E. Grossmann).
1 Introduction

Optimization of mixed discrete-continuous systems has advanced greatly in recent years because of increasing computing power and algorithmic improvements. Both of these enabling technologies develop unabated. As a result, classical control engineers are augmenting their systems with discrete dynamics and traditional computing theories are incorporating timing constraints, narrowing the gap between the continuous and discrete mathematics communities. One consequence is that we are considering systems of increasing complexity, not only in terms of computational tractability but also in the number of features considered, the scale of the systems, and the number and types of algorithms used on the systems. This creates several modeling challenges: intuitive mathematical constructs are needed to represent the diverse systems considered, models should be modular, and it should be possible to apply various algorithms to the same model. Network models with switched flows elucidate these issues and represent a class of systems of practical importance. We present a new formal modeling framework, based on hybrid automata, for such systems and provide systematic transformations to existing frameworks, enabling use of known solution methods across disciplines.

Our central concern is to ease the burden of modeling, especially for optimization of hybrid dynamic systems. Formulating models is laborious and constitutes a major portion of engineering costs. Roughly the workflow of engineering involves: gaining an understanding of a system and the problem to be solved, creating a conceptual representation of these, defining the variables of the system, declaring a formal model involving those variables, designing an algorithm to solve the problem, and finally computing the solution. A conceptual representation is a description formulated using English (or your preferred language) and pictures. Systems are often viewed as networks and, in the case of optimization, alternative choices are sometimes viewed as a superstructure, as in Yeomans and Grossmann (1999). The first step in formalizing the conceptual representation is to declare the variables and specify their types. Then we create a formal model, an unambiguous description of a system formulated in a mathematical language. Given a model, algorithms are designed to solve a problem involving the model. The vast majority of research in optimization is focused on this last step. Creation of a model is taken for granted, but, as emphasized in von Wedel, Marquardt, and Gani (2002) and Zentner, Elkamel, Pekny, and Reklaitis (1998), modeling challenges are a serious obstacle that must be addressed.

The scope of physics considered includes piecewise linear dynamics in the continuous domain and discrete dynamics governed by a variety of logical constraints.
Strictly, such a system can be represented as a mixed integer program (MIP), currently the most general purpose framework for optimization. Johnson et al. (2000) and Bixby (2002) review impressive advances in this area that are allowing increasingly larger problems to be optimized, and Kallrath (2000) discusses its importance in the process industry. However, the modeling language used by MIP is restrictive, and in practice it can be very difficult to represent even simple systems. Numeric data types, real and integer, with their associated operations are the only constructs supported. Constraint logic programming, Van Hentenryck (1989), provides additional data types but still does not address many of the needs in process engineering.

We address this problem by formulating a new modeling language, which more closely represents natural thoughts. The primary construct is the component automaton, a construct similar to other hybrid automata definitions in the literature. The complete model is a set of component automata coupled by overall system constraints. This model is extended to a network model which maps more closely to physical systems by defining flow relationships and material conversions. Finally, these models are converted into MIP models using a sequence of transformations: the discrete dynamics of automata are restated as disjunctions, statements in finite domain logic are converted into Boolean logic, variable arguments are removed, and quantifications over reals are made finite. The conditions under which a transformation to a MIP is possible are maintained from the onset. The net result is an improved modeling system tied to existing algorithms.

1.1 Modeling Challenges

Example: We wish to convert the following conceptual description into a formal model.

A reactor can run one of three reactions. If rxnX is run the temperature must stay below 400. If rxnZ is run the temperature must be below 600. There is no constraint for rxnY, but the maximum temperature the reactor can handle is 1000.

Many models can be provided. A possible MIP model is given on the left in Table 1, and an improved formulation, in the sense that it is easier to declare and understand, is provided on the right. Variables in the logic model are physical; both $\Theta$ and $Q$ represent real concepts. The MIP model requires unphysical variables; $\Theta_x$ is the temperature of the reactor for reaction X, and there are also
other temperatures. Unphysical variables and unintuitive constraints come together. The constraint $\Theta = \Theta_x + \Theta_y + \Theta_z$ says that the reactor temperature is a sum of three other temperatures, an awkward statement of physics which makes sense only when the other MIP constraints are considered simultaneously. In contrast, the constraints of the logic model are self-contained and virtually identical to the conceptual description; English words are simply replaced by shorter mathematical symbols. Finally, the logic model is smaller; it requires 2 variables and 3 constraints as compared with 7 variables and 5 constraints in the MIP model. The new data type $Q$ and logic operators make the new model easier to declare, understand, and extend.

<table>
<thead>
<tr>
<th>MIP</th>
<th>Finite Domain Logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Theta, \Theta_x, \Theta_y, \Theta_z \in \mathbb{R}$</td>
<td>$Q = {\text{rxnX, rxnY, rxnZ}}$</td>
</tr>
<tr>
<td>$x, y, z \in {0, 1}$</td>
<td>$\Theta \in \mathbb{R}, \quad Q \in Q$</td>
</tr>
<tr>
<td>$x + y + z = 1$</td>
<td>$Q = \text{rxnX} \implies \Theta \leq 400$</td>
</tr>
<tr>
<td>$\Theta = \Theta_x + \Theta_y + \Theta_z$</td>
<td>$Q = \text{rxnZ} \implies \Theta \leq 600$</td>
</tr>
<tr>
<td>$\Theta_x \leq 400x$</td>
<td>$\Theta \leq 1000$</td>
</tr>
<tr>
<td>$\Theta_y \leq 1000y$</td>
<td></td>
</tr>
<tr>
<td>$\Theta_z \leq 600z$</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Two formal models for a single conceptual model.

This example demonstrates the mismatch between conceptual and formal models, the first modeling challenge addressed. The conceptual description has no mention of 0-1 valued numbers, but these are the only data types allowed in a MIP. No statement is ever made on what a reaction is doing, but that is what the variables $x$, $y$, and $z$ represent. The statement regards the reactor’s state, which is exactly what $Q$ represents. The gap between the conceptual and formal model is narrowed by introducing new data types and operations into the mathematical language. The result is more natural and compact models.

Raman and Grossmann (1994) discuss how disjunctive programming (DP) simplifies modeling for similar reasons. DP includes Boolean variables and “disjunctions”, which represent a choice of different constraints dictated by logical conditions. DP models can be transformed into MIP models, but the modeler need only employ the richer language, leaving the transformations to the specialists. Vecchietti and Grossmann (1999) discuss a software implementation of these kinds of transformations. In the example, the implications can be stated as disjunctions. Then the convex hull method gives the stated MIP.
Discrete dynamics pose a second challenge. Hooker (2000) discusses several classes of logic models used for optimization. Goldblatt (1992) presents various temporal logics which add a notion of time. In process systems, continuous and discrete dynamics are often combined in the style proposed by Barton and Pantelides (1994) and the mixed logical dynamic (MLD) system of Bemporad and Morari (1999). Heemels et al. (2001) demonstrated the equivalence of MLD with several other modeling classes, and some of the transformations have been implemented in software described in Potocnik et al. (2003). Perhaps the most widely used formalism is the hybrid automaton discussed in Henzinger (1996). Their model makes use of the finite domain data type and is an intuitive representation.

Hybrid automata also make explicit the notion of time, and that events occur in a sequence as time progresses. The conceptual understanding of a discrete dynamic system surely includes statements such as “if something happens, then later something else might or does happen”. MIP style formulations require flattening the progression of time, negating the intuition.

MIP formulations also imply that an optimization problem is the problem to be solved. The model is designed to answer only one question. But certainly other problems are also of interest, such as feedback control, simulation, and verification. In addition, for this single problem, the MIP format is restricted to a small number of solution strategies. For example, constraint programming (CP) algorithms often handle logical constraints for scheduling better than those for MIP. Recent results in Hooker (2000), Jain and Grossmann (2001), Van Hentenryck (2002), and Maravelias and Grossmann (expected 2004) suggest that combining MIP and CP methods sometimes gives solutions faster than either method alone. The proposed modeling framework is independent of the solution method and transformations can be provided to various formats.

The third modeling challenge addressed is the ability to represent large-scale systems. That a modeling system must support modularity is so basic a notion that it has long been done in the continuous domain without explicit mention, but the dynamics of a hybrid system pose unique challenges. Traditional process flowsheets, Westerberg (1979), allow declaring unit operations and combining them into an overall plant model. Flowsheets are a type of network, which are frequently used to ease modeling, e.g. Westerweele et al. (1999) and Gilles (1998). Usually, each node is spatially distinct. The state-task network model of Kondili, Pantelides, and Sargent (1993) extends this by modularizing not just by space (the state) but also by function (the task), introducing discrete variables in the process.
The logical limit of this is to create modules, each of which govern their own variables, real and discrete. Then provide overall system constraints which are a function of all the variables from each of the modules. A particular formulation of a hybrid automaton, a component automaton, will be introduced, and we discuss how it serves as a very general module. Then a complete system can be described by a set of coupled component automata. Additional constraints are provided by superimposing the component automata onto a network structure. The network serves to restrict flow relationships and guarantees mass conservation.

1.2 Application Domains

Much of the background on hybrid automata comes from computing systems, traditionally studied with strictly discrete mathematics. The need for continuous variables became apparent when researchers considered computers interacting with natural phenomena. Alur and Dill (1994) define a timed automaton, in which continuous variables vary at rate 1 and are sometimes reset to 0, allowing real-time constraints to be considered in computing systems. Cassez and Larsen (2000) extend this by allowing the rate of change to be 0 or 1 and call this a stopwatch automaton. The model we present is a further extension; rates can be chosen from an arbitrary set of real values and is similar, in regards to algorithmic complexity, to the piecewise constant derivative automaton of Asarin et al. (1995).

Optimal paths for the timed and stopwatch automaton are provided by Alur et al. (2001) and Abeddaim and Maler (2001), respectively. Their methods exploit special features resulting from the restricted continuous dynamics. Our strategy is to translate to a mathematical program (MP), allowing use of the well developed methods from that area. Stursberg et al. (2002) and Panek et al. (2003) propose a similar scheme for different automata models.

The STN model of Kondili et al. (1993) was one of the first attempts to incorporate general networks and multiproduct flows in the scheduling and planning of process systems. It and its extensions use batch processing, which is modeled by instantaneous flows. Thus, real variables’ values jump at event times but are otherwise constant. This can be a useful approximation and allowed embedding discrete decisions into plant models. However, true continuous dynamics are not incorporated.

Switched flow networks are the application domain discussed in this paper. Figure 1 depicts part of a flow network. Three materials are produced at $v_5$, each by a different reaction, but with only one reactor to use. Figure 2 depicts the discrete modes and the possible transitions between them. In each mode, materials
flow in/out and are consumed/generated at some constant rates. Various logical conditions can force, allow, or prohibit discrete transitions. Transitions may be possible but not desirable, depending on the objective. No production occurs during setup and there may also be a setup cost; thus minimizing switching may be preferred. This means the sequence in which reactions are run can be important. Finally, the material levels at $v_4$ are also dependent on upstream processes, so it may be desirable to run upstream processes first.

![Flow network](image1)

**Figure 1:** Flow network.

![Discrete modes](image2)

**Figure 2:** Discrete modes for $v_5$.

This system is distinguished from much of the scheduling and planning literature in Operations Research (OR), where the focus is on discrete parts flows. Baptiste, Le Pape, and Nuijten (2001) recently reviewed constraint based scheduling methods. There, a certain integer number of jobs must be processed on various machines for certain amounts of time. The process industry deals with fluid products, so the metric is a real valued quantity of material processed, and each can be processed at certain rates on each machine. The switched flow network can be viewed as a fluid version of classic job shops. Forrester’s (1961) seminal work actually “suggest[s] that the system be treated, at least initially, on the basis of continuous flows”. His qualification “at least initially” implies that he considered this an approximation, a viewpoint that is frequently repeated in the OR literature. For the process industry, fluid flow is the more accurate model.
2 Preliminaries

To introduce the proposed modeling framework, we first define a timeline suitable for hybrid systems. This will be followed by an introduction to data types and the operations which can be performed on those types. The end result will be the ability to specify classes of constraints, distinguished by the data types and variables they involve. These constraint sets and the timeline will be integral to the full modeling framework to follow.

2.1 Hybrid Timeline

For continuous systems, the timeline is simply an interval of the real numbers. However, discrete dynamics occur instantaneously and the timeline must allow specification of two values at certain time points, called event points. This will also allow the specification of discontinuities in continuous variables.

Figure 3 depicts a hybrid timeline, defined by Lygeros et al. (1999) as an ordered set of intervals $T = \{I_i\}_{i=1}^n$ such that

- $I_i = [t_i^s, t_i^e]$ for $i < n$ and if $n < \infty$ then $I_n = [t_n^s, t_n^e]$ or $I_n = [t_n^s, \infty)$;
- for all $i$, $t_i^s \leq t_i^e = t_{i+1}^s$.

The interpretation is that all discrete variables are constant during each interval. We reformulate the above definition into one more useful for our purposes. Define a hybrid timeline $T = \{(i, t) : t \in [t_i^s, t_i^e] \in \hat{T}\}$. With this definition, a single time point is a pair $(i, t)$. As expected of a timeline, there is a strict order relation $<$. The time point $\tau = (i, t)$ precedes $\tau' = (i', t')$, denoted $\tau < \tau'$, if and only if $i < i'$ or $t < t'$. If $\tau \neq \tau'$ and $\tau \neq \tau'$, then it must be that $\tau' < \tau$. Let $\mathbb{N}$ denote the set $\{1, 2, \ldots, n\}$ and $\Delta t_i = t_i^e - t_i^s$ the length of interval $i$.

![Hybrid timeline diagram](image)

Figure 3: Hybrid timeline.

The set of event points is $\hat{T} = \mathbb{N} \setminus \{n\}$ if $n < \infty$ or $\hat{T} = \mathbb{N}$ otherwise. (The timeline will eventually be bounded to limit the search space of the optimization...
problems.) Each event point coincides with two time points: \( \tau = (i, t_i^e) \) and \( \tau' = (i + 1, t_{i+1}^e) \). Only the integer component of time changes when an event occurs. The real component at both times is \( t = t_i^e = t_{i+1}^e \).

Time is usually considered continuous or discretized, but additional possibilities exist in a hybrid system. The formulation provided is a continuous timeline with event points. If \( n = 1 \), then we have a continuous time formulation without events. If \( n > 1 \) and interval lengths are fixed, the system restricts event points to a finite set of times. However, there could still be continuous evolution within an interval. Finally, time could be discretized within each interval, whether or not interval lengths are fixed, to accommodate solution of differential-algebraic equations (DAEs). In summary, a hybrid timeline can be discretized or continuous and can either restrict event times to fixed points or not.

### 2.2 Data Types and Dynamic Variables

The *type* of a variable is defined by the set from which it can take values. For example, variables of type real can be assigned a value from the set \( \mathbb{R} \). Also associated with a type is the operations defined on it. The operation `+' takes two reals and returns a real, i.e. \(+ : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R} \). The blackboard bold font, e.g. \( \mathbb{R} \), is used to denote the space from which a variable takes values; normal font, e.g. \( X \), is used for variable names; and bold font, e.g. \( \textbf{X} \), denotes a set of variables.

A finite domain variable, sometimes called set valued, takes values from a finite set \( \mathbb{Q} \). This is a user defined type because the set \( \mathbb{Q} \) can be defined as needed. For the introductory example, the space is \( \mathbb{Q} = \{\text{setup}, \text{rxnX}, \text{rxnY}, \text{rxnZ}\} \). Now a variable \( \mathbf{Q} \) of type \( \mathbb{Q} \) can be defined to give the discrete mode of the system. Actually, this is a family of types because many such sets might be used. Each type in this class must be distinguished, thus \( \mathbb{Q}_1, \mathbb{Q}_2, \ldots \) denote different types. Usually, only a single variable of each finite domain type is needed. Thus, \( \mathbb{Q}_1 \) and \( \mathbb{Q}_2 \) are understood to be variables of type \( \mathbb{Q}_1 \) and \( \mathbb{Q}_2 \). Elements of a finite domain are named, but the names carry no mathematical meaning, e.g. \( \text{stateA} \in \mathbb{Q}_1 \) is unrelated to \( \text{stateA} \in \mathbb{Q}_2 \), just as 5.0 kg is unrelated to 5.0 Pa.

Sets are used in MP, but they are not a datatype because variables can not take on values from this set. Sets in MP serve only as indices and are merely a syntactic convenience, albeit a major one. CP does allow set valued variables.

Sets of variables are used instead of vectors and matrices. A set of real variables is denoted \( \mathbf{X} \) and \( \mathbf{X} \) denotes an element of the set. Another set \( \mathbf{y} \) of real variables might also be defined with \( y_1, y_2, r, s, \ldots \) as elements. Similarly, \( \mathbb{Q} \) is a set of finite domain variables. Collectively, \( \mathbf{V} \) denotes a set of variables of any type, e.g.
Finally, tuples are used to provide aggregate descriptions of complex models. The tuple $T = (A, B, C)$ has three elements, each of which can be anything, a variable, a set, or another tuple. Once a tuple is defined, multiple instances of it may be needed, just as multiple real variables are needed. Superscripts will be used to distinguish several instances of the same type of tuple, e.g. $T^\alpha = (A^\alpha, B^\alpha, C^\alpha)$ and $T^\beta = (A^\beta, B^\beta, C^\beta)$.

A dynamic variable is a function $v : T \rightarrow V$, where $V$ is the set associated with $v$’s type. A continuous variable changes infinitesimally in an infinitesimal amount of time, but also its value can jump instantaneously. Within an interval, differential equations define the dynamics of real variables. At an event point $i$, which recall starts at $\tau = (i, t)$ and ends at $\tau' = (i + 1, t)$, the real value of time does not change but the integer component does. Thus, changes at an event point are provided by algebraic constraints involving the variable evaluated at the two times, e.g. $X(\tau') = X(\tau) + 1$ would increment the value of $X$ by 1 at event $i$. When necessary, the time points will be expanded, so the previous equation would be written $X(i + 1, t) = X(i, t) + 1$.

A dynamic finite domain variable $Q : T \rightarrow Q$ takes on values from a finite set and can not evolve continuously. Its value changes only at event points. The automaton and finite domain logic, explained subsequently, provide methods for specifying discrete dynamics. Since a discrete variable is constant within intervals, it suffices to make it a function of the event points, $Q : N \rightarrow Q$ with $Q(i, t) = Q(i)$ for all time points.

### 2.3 Logic Formulae

A Boolean logic formula is one which evaluates to an element of the Boolean space $\mathbb{B} = \{\text{true, false}\}$, not to be interchanged with the binary $\{0, 1\}$ space. Equations will also interpreted as logic formulas. When we write $X = 5$ we mean to find the value of $X$ such that the statement is true. Thus, equations are the function $=: \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{B}$. Instead of a set of equations, we say there is a single Boolean formula which is the conjunction of several equations. Let ‘\&’ denote conjunction; note $\land : \mathbb{B} \times \mathbb{B} \rightarrow \mathbb{B}$. Then,

\[
\begin{align*}
X_1 + 2X_2 &= 3 \\
3X_1 + X_2 &= 4
\end{align*}
\]  

is interpreted as $(X_1 + 2X_2 = 3) \land (3X_1 + X_2 = 4)$. One can think of the invisible new line symbol as an alternative for ‘\&’.
All data types have at least one logic operation defined for them, that of equality. Consider a discrete constant \( c \in \mathbb{Q} \), just as \( 3.14 \in \mathbb{R} \) is a real constant. The variable \( Q \) of type \( \mathbb{Q} \) might be involved in a constraint \( Q = c \), which is either true or false depending on the value of \( Q \). Now, it becomes valid to declare statements mixing reals and finite domain types, e.g. \((X_1 = X_2) \land (X_2 \geq X_3) \land (Q = c_1 \lor Q = c_2)\).

These concepts are defined precisely in the appendix. Given a set of real variables \( X \) and discrete variables \( Q \), and their union \( V = X \cup Q \), we define \( \mathcal{L}(V) \) to be the set of all valid formulas involving the variables in \( V \). For example, the logic formula (1) is an element of \( \mathcal{L}(V) \). \( \mathcal{L}(X) \) contains only statements on real variables because no finite domain variables are in \( X \). The definition guarantees that only linear constraints are included. Thus, \( \mathcal{L}(X) \) is the set of all constraint sets used in linear programming (LP) models. Let \( X(\tau) \) denote the values of a set of real variables at time \( \tau \). Then, \( \mathcal{L}(X(\tau)) \) denotes the set of linear relations evaluated at time \( \tau \). Other classes of constraints can be declared compactly as needed. \( \mathcal{L}(Q) \) is the system of finite domain logic and \( \mathcal{L}(V) \) is one formulation of a hybrid system, although not the complete modeling framework proposed. As a well defined set, \( \mathcal{L}(V) \) can itself be treated as a space involved in function definitions. For example, \( g : \mathbb{Q} \rightarrow \mathcal{L}(V) \) is a function valued variable that associates a constraint with each mode.

3 Modeling Hybrid Dynamic Systems

\( \mathcal{L}(V) \) is a modeling language comparable to MIP. Instead of binary variables, it provides the richer finite domain logic. However, it does not explicitly support dynamics, modularity, and conservation of mass. Additional modeling classes will now be presented, each building on the previous. Figure 4 depicts the relationships between the models we will introduce. The component automaton \( A \) governs the dynamics of a single discrete variable and affects the continuous variables in a manner dependent on its discrete state. Model \( M \) will be comprised of several automata and constraints which couple their behavior. A differential and discontinuity equation take their parameters from the automata. Finally, model \( N \) will be built by adding a material flow network and reactions. Each extra feature provides the modeler with additional forms of expression. These features closely match conceptual representations and lead to more intuitive formal models.
3.1 Component Automata $\mathcal{A} = (Q, x, \hat{x}, Inv, E)$

The component automaton is so named because its utility is as a component of a larger system. It defines the discrete modes of a process and how the process affects some overall system variables, in each mode and when transitioning from mode to mode.

An example will help to introduce terminology before we provide a formal definition. Figure 5 depicts an automaton with two modes, or discrete states. There is a set of rate variables $\{m, p, c\}$ which take different values in the different modes. In mode on, the process is filling a tank at rate $m = 3.2$, and an operational cost is incurred at rate $p = 1.0$. The invariant, or path constraint, $M(\tau) \leq 100.0$ requires that the tank level be below its maximum capacity. The automaton can not be in the on mode when this condition is violated. There is no such requirement in the off mode. When transitioning to off a reset forces the value of a clock to 0.0. While in the off mode, the clock variable evolves at rate $c = 1.0$, i.e. it measures elapsed time. The guard $C(\tau) \geq 15.0$ must be satisfied to transition from off to on, so the automaton must stay in the off mode for at least 15.0 time units each time it enters that mode. There is also a reset on this transition, a setup cost of $^{\text{\hat{p}}} = 10.0$ units is incurred whenever this event occurs.

The differential equation governing $M(\tau)$ has not yet been specified. The automaton described is just one of several that may be affecting the tank level. In the off mode, this process is not adding or removing any material from this tank, but another process might be. The overall rate of change of $M(\tau)$ will be dependent on a set of component automata. Similarly, $p$ is just the rate at which costs are incurred due to this process. The overall cost incurred is $P(\tau)$. The jump variable $\hat{p}$ increments the value of $P$ when certain events occur, e.g. by 10.0 when transitioning from off to on. At the same event time, other processes may also increment the value. The set of jump variables is $\{\hat{m}, \hat{p}, \hat{c}\}$ but $\hat{m}$ is not in
any resets, i.e. this process does not make any instantaneous changes to material levels.

We now provide the general definition for a component automaton. Assume there exists a system (defined in the following section) with a timeline $T$ and a set of real valued variables $X$. In the context of such a system, the component automaton

$$A = (Q, x, \dot{x}, Inv, E),$$

where $Q : \mathbb{N} \rightarrow \mathbb{Q}$ gives the discrete state in each interval; $x$ is a set of rates dependent on the state—each is given by $x : Q \rightarrow \mathbb{R}$; the set of jump variables $\dot{x}$ are defined at the event points—each $\dot{x} : T \rightarrow \mathbb{R}$; the invariant condition $Inv : Q \rightarrow L(X(\tau))$ is imposed as $Inv(Q(i)), \forall \tau \in T$; and the set of transitions is $E \subset Q \times L(X(\tau)) \times L(X(\tau) \cup \dot{x}(i)) \times Q$. Each $(q, \gamma, \rho, q') \in E$ defines a discrete transition from mode $q$ to mode $q'$. The transition can be made at event point $(\tau, \tau')$ only if the guard $\gamma$ holds true at time $\tau$, and the reset\(^1\) $\rho$ is also enforced if the transition is made. There exists at most one transition from any $q$ to $q'$. A dummy transition $(q, \gamma, \rho, q)$ is required for every $q \in Q$, with $\gamma = \text{true}$ and $\rho = \land_{x \in \dot{x}}(\dot{x}(i) = 0)$. The motivation behind these definitions will become clear in the next section, when we define a complete modeling system.

Component automata have a convenient visual representation, as shown in Figure 5. Draw a box for each discrete state. Within the box, put the name of the mode at the top, followed by the values of the rates in that mode, and the invariant for that mode at the bottom. Add an arrow from mode $q$ to $q'$ if $(q, \gamma, \rho, q') \in E$. Write the guard $\gamma$ for this transition near the tail of the arrow.

\(^1\)Strictly, resets $\rho \in L(X(\tau) \cup \dot{X}(\tau') \cup \dot{x}(i))$ where $\dot{X}$ is a set of variables known to be dependent only on the current automaton. For example, $C'$ is used in a reset in figure 5.

Figure 5: Tank on/off flow.
and write the reset $\rho$ near the arrowhead. Clutter is reduced by not showing zero values: when the value of some $x \in \mathbf{x}$ is not shown in a mode, it is understood to be zero, and if some $\dot{x} \in \dot{\mathbf{x}}$ does not appear in $\rho$ for a transition, the term $\dot{x}(i) = 0$ is implicitly included. Since this precludes the possibility of leaving some $\dot{x}$ unconstrained, let “$\dot{x}(i)$ free” denote that $\dot{x}(i) = 0$ should not be automatically included. When a guard or invariant is not shown, its value is true. Dummy edges need not be shown because their definition is fixed. Thus, Figure 5 is a formal model.

### 3.2 Coupled Component Automata $\mathcal{M} = (T, X, Q, F)$

A complete model for a hybrid dynamic system includes a set of component automata. Such modular synthesis provides tractable modeling capabilities. Consider the tank filling example of Figure 5, and let us call this automaton $\mathcal{A}_\alpha$. The rates and jump variables of that automaton will also now be denoted with a superscript $\alpha$. Suppose there is another process $\mathcal{A}_\beta$ also adding and removing material from the same tank. The tank level $M$ is an overall system variable. Unlike rates and jump variables, it is not part of an automaton. The net rate of change in the tank level would be $\dot{M}(\tau) = m^\alpha + m^\beta$, a value which switches depending on the modes of both automata.

We now define our first complete modeling framework for hybrid systems and discuss how it handles discrete interactions without loss of generality. The coupled component automata model is

$$\mathcal{M} = (T, X, Q, F),$$

where $T$ is the hybrid timeline; $X$ is a set of real valued system variables; $Q$ is a set of component automata and also denotes the set of discrete variables because there is exactly one discrete variable for each automaton; and an overall system constraint $F \in \mathcal{L}(\mathbf{V}(\tau))$, where $\mathbf{V} = \mathbf{X} \cup \mathbf{Q}$, must hold at all times regardless of the discrete modes of the automata. Constraints at specific times are also allowed, initial conditions for example.

$F$ can be thought of as a constraint set which allows specifying additional conditions not defined within the component automata. Especially important are constraints that couple the automata. In particular, the following constraints (2) and (3) define the dynamics of $X$ and are always included in $F$. Each $X \in \mathbf{X}$ is a function of some variables from $\cup_{\alpha \in \mathbf{Q}} \mathbf{x}^\alpha$. Each lower case $x^\alpha \in \mathbf{x}^\alpha$ is the contribution of $\mathcal{A}_\alpha$ to the overall continuous evolution of the upper case $X$. 

14
Precisely, this is done by the differential equation

$$\dot{X}(\tau) = \sum_{\alpha \in Q} x^\alpha(Q^\alpha(i)) + k \quad \forall \tau \in T$$

(2)

where the dot in $\dot{X}(\tau)$ denotes differentiation with respect to the second component of $\tau$ and $k$ is some constant. Thus, the overall rate of change of $X$ is potentially dependent on the active modes of all automata. Integration across event points is not possible, but within each interval

$$X(i, t) = X(i, t_i) + \int_{t_i}^t \left[ \sum_{\alpha \in Q} x^\alpha(Q^\alpha(i)) + k \right] dt \quad \forall (i, t) \in T.$$ 

The integrand is constant within the bounds, leaving an algebraic expression of the form allowed for $F$. Continuous evolution is dictated by $x$, and, analogously, discontinuities in $X$ are given by $\dot{x}$. Jumps in each $X$ at event points are given by the discontinuity equation

$$X(\tau') = X(\tau) + \sum_{\alpha \in Q} \dot{x}^\alpha(i) \quad \forall i \in T$$

(3)

where, recalling the notational convention, the $i$th event point occurs from time $\tau$ to $\tau'$. In addition to the differential and discontinuity equations, $F$ may include initial conditions, universal constraints such as nonnegativity of mass, and other coupling constraints as needed for a particular problem.

### 3.2.1 Discussion of $\mathcal{M}$

Unfortunately, the interaction of automata is complicated by the fact that a single continuous variable is affected by multiple discrete events. Consider that on some transition of $\mathcal{A}^\alpha$, the tank level $M$ is reset to zero and on some transition of $\mathcal{A}^\beta$, $M$ is not reset. Now, if those two transitions occur simultaneously, what should the value of $M$ be? We could require that those two transitions not occur simultaneously. However, that is unlikely to have been the intended meaning.

Also, say $\mathcal{A}^\alpha$ is in some discrete state and according to it, $M$ should be evolving at some constant rate. However, $\mathcal{A}^\beta$ makes a transition and wants to reset $M$ to zero. At what time, in the hybrid timeline, does this occur?

The component automaton has been designed to address these issues. In fact, the discrete dynamics do not directly affect the continuous system variable $M$,
as supposed in the above arguments. Equations (2) and (3) affect $X$ via $x^a$ and $\dot{x}^a$ for all automata. Thus, each automata $A^a$ dictates the values of only its own variables $x^a$ and $\dot{x}^a$. Continuous evolution of $X$ is defined by $\dot{X}$, and values of $\dot{X}$ are not directly specified by the automata. Also, the values after a transition are $X(\tau')$, and resets do not involve this variable; they may only refer to the value at the start of the transition $X(\tau)$. So in discrete evolution also, there can be no conflict in multiple automata trying to set a variable to different values.

The second issue discussed above regards how to define the system when one automaton transitions but another does not. By using a single hybrid timeline, it has been implicitly declared that automata transitions occur synchronously. By itself, this would be an excessive and arbitrary restriction. However, the dummy transitions effectively provide asynchronous semantics. A dummy transition allows an automaton to progress in time but in a way that has no consequence. The discrete state does not change, the transition is always allowed and never forced, and all $\dot{x}$ are 0, thus making no change to continuous variables. At event points, an automaton can simulate staying in a state by making a dummy transition.

Approximations which reduce computational complexity are now easily stated. Discretized time formulations require two restrictions: no continuous evolution and a fixed step size. The former is attained by setting all rates to zero, $x^a(q) = 0$ for all $x^a \in x^a, q \in Q^a, a \in Q$. If one wishes to allow discrete evolution at every time point, as in Perea-Lopez et al. (2001), then every time point corresponds to an event point. Thus, set $\Delta t_i = k$, where $k$ is the step size. Appropriate specification of resets then allows the discretized approximation to continuous evolution as desired in such systems. Leaving off the step size constraint gives continuous time formulations entailing models of the form in Maravelias and Grossmann (2003).

An essential contribution of the cited models is the particular formulation the authors devised to improve computational times. The proposed modeling framework is more general in the sense that special cases of it are semantically equivalent to those cited. $\mathcal{M}$ is an abstraction designed to provide the semantics of the system in the simplest and most general manner possible. Selecting amongst equivalent formulations is part of the solution process. Higher level modeling languages such as $\mathcal{M}$ facilitate solution methods by leaving bare the problem structure.

3.3 Network of Hybrid Systems $\mathcal{N} = (\mathcal{M}, G, P, R, \phi)$

Real systems are often perceived schematically, with materials flowing from point to point. Model $\mathcal{M}$ does not support such a feature. In addition, mass conserva-
tion is not guaranteed as part of the system definition. Rather, it must be assured by the modeler carefully defining equations (2) and (3) for each $X$ representing a mass variable. Network models address these issues and provide an intuitive representation for the relationships between different processes. Each node in a network defines material holding points and processes define the flows between them. There are usually a set of materials flowing through the network. Material conversions are any change in thermodynamic state and include reactions, heating, and separation, and also assembly and disassembly under continuous approximations. Without material conversions, the network defines a transportation system.

Consider again the example of Figure 5. In the on mode, this process is putting material into a tank at rate $m$. Presumably, this material is coming from somewhere. So we must define another rate specifying the removal of material from another vessel and set its value equal to the negative of $m$. If the tank was instead a reactor, there would be several consumption and generation rates, each related to one another by a reaction rate and stoichiometric coefficients. It would be better to define the single reaction rate than numerous other rates which are all closely related—the latter is more taxing and error prone. Finally, given that all input and output flow rates and generation/consumption rates are specified, the mass balance equation can be inferred.

We now define the constructs required to support these features. A network of hybrid systems is

$$
\mathcal{N} = (\mathcal{M}, G, P, R, \phi),
$$

a coupled component automata model $\mathcal{M}$ augmented with a network $G = (V, A)$, where $V$ is a set of nodes and $A \subseteq V \times V$ are directed arcs; $P$ a set of materials; $R$ a set of material conversions; and $\phi : R \rightarrow V \times Q \times P$ gives the node and rates at which each material conversion is running. The additional constructs are natural; conceptual representations often include a picture of a network and reactions are thought of before the differential equations they imply. We take these constructs out of the undefined conceptual space and make them precise. Flow networks and symbolic reaction definitions such as $A + B \rightarrow X$ become well defined constraints.

A network model specifies levels of each material at each node. Thus, a subset of $X$ are mass variables of the form $M : P \times V \rightarrow \mathbb{R}$, the level of material $p$ at node $v$. Flow relationships require that material flowing into a node is coming from another node. In other words, flow rates are indexed by the arcs, $m_{puv}$ is the flow rate of material $p$ from node $u$ to $v$. By convention, flow rates are nonnegative—
they represent the flow along an arc in one direction, not the net flow. Finally, material is generated at some node by one of the material conversions in $R$. Thus, generation rates should be of the form $g_{q_{pv}}$, the rate at which material conversion $\zeta$ generates (negative value implies consumption) material $p$ at node $v$. Similar discussions apply to $\dot{m}_{p_{uv}}$ and $\dot{g}_{q_{pv}}$.

The requirement on the user is that she specifies the materials $P$ and makes sure that mass levels, flows, and generation rates are correctly defined as functions of this set. This is always done anyway but only informally. Optimization software do not allow declaring a set to specifically be a set of materials. Doing so allows automatic generation of statements of mass conservation, nonnegativity of mass, and various flow constraints, as will be shown subsequently.

Each material conversion $\zeta \in R$ is a specification of stoichiometric coefficients, $\zeta : P \rightarrow \mathbb{Z}$. A species $p$ with $\zeta_p < 0$ is a reactant, with $\zeta_p > 0$ is a product, and $\zeta_p = 0$ means the material is not involved in the material conversion. Stoichiometric coefficients are simply $\pm 1$ for heating but the definition also allows reactions and separation. $\phi(\zeta) = (v, \alpha, b)$ means $\zeta$ is running at node $v$ and the conversion rates are $r^\alpha$ and $\dot{r}^\alpha$ with respect to species $b$. Note $r^\alpha$ and $\dot{r}^\alpha$ have dimensions of mole/time and mole, respectively. It will be convenient to denote material conversions as

$$\sum_{p: \zeta_p < 0} -\zeta_p P \rightarrow \sum_{p: \zeta_p > 0} \zeta_p P.$$  \hspace{1cm} (4)

Since $r^\alpha \in x^\alpha$, the reaction rate is mode dependent. For example, a reaction may be turned on or off, or may be run at different rates under different heater settings. Discretized models which assume that reactions take place instantaneously are accommodated by giving nonzero values for $\dot{r}^\alpha$. Constant conversion rates are not accommodated in the definition, but as with fixed flows, fixed material generation rates can be absorbed into the constant $k$ of equation (2).

The network and material conversions of model $\mathcal{N}$ are intuitive constructs and replace numerous equation declarations. They also provide more information on the nature of the problem, allowing automatic guarantees of mass conservation in a mixed continuous-discrete event system. Our main example will elucidate these benefits.
4 Optimization Problem on $\mathcal{N}$

A model $\mathcal{N}$ is not a deterministic system\(^2\), i.e. there are many feasible trajectories. There is freedom to choose the length of time spent in each discrete mode and various discrete transitions are possible. Let $\Xi_\mathcal{N}$ be the set of all feasible trajectories. This is a subset of the full space $\mathcal{T} \times \mathbb{R}^{|\mathbf{X}|} \times \mathcal{Q}_1 \times \cdots \times \mathcal{Q}_j$, the continuous and discrete values at each point in the timeline. An objective $\Omega$ is a metric on this space, and an optimization problem seeks a solution in $\Xi_\mathcal{N}$ such that the metric is minimized (or maximized). Thus, the optimization problem is

$$\min_{\xi \in \Xi_\mathcal{N}} \Omega$$

s.t. $\mathcal{N} = (\mathcal{M}, G, P, R, \phi)$. \hspace{1cm} (PA)

It consists of three definitions: a model, an objective, and a feasible space.

We consider objectives that are a function of the event times and the values of continuous variables at those times. For example, one may wish to optimize the final value of a finite run,

$$\Omega = X(\mathcal{N}, t_{N}^e).$$ \hspace{1cm} (5)

Or often, one is concerned with a time average performance criterion,

$$\Omega = \frac{1}{(t_{N}^e - t_{i}^e)} \int_{t_{i}^e}^{t_{N}^e} X(\tau) \, dt$$

$$= \frac{1}{2 (t_{N}^e - t_{i}^e)} \sum_{i \in \mathcal{N}} (X(t_{i}^e) + X(t_{i}^e)) \Delta t_{i}$$ \hspace{1cm} (6)

where the integral calculates trapezoidal areas because continuous variables evolve piecewise linearly. Finally, in makespan problems, the value of time is itself the objective,

$$\Omega = t_{N}^e.$$ \hspace{1cm} (7)

4.1 Hybrid Trajectories

It remains to define the feasible space $\Xi_\mathcal{N}$. A single trajectory is a mapping from a timeline to the set of possible values for all state variables. View the system

\(^2\)We refer to $\mathcal{N}$ in this section, but everything applies equally to $\mathcal{M}$ because both have the same state variables.
as progressing in time with alternating occurrences of discrete and continuous evolution. A continuous trajectory is an interval

\[ [(\tau, \mathbf{X}(\tau), \mathbf{Q}(\tau)) \rightarrow (\tau'', \mathbf{X}(\tau''), \mathbf{Q}(\tau''))] \]

where \( \tau \) precedes \( \tau'' \) and—letting \( \tau' \) be any time point such that \( \tau < \tau' < \tau'' \)—the interval does not change, i.e. the integer components of \( \tau, \tau' \), and \( \tau'' \) are equal, which implies \( \mathbf{Q}(\tau) = \mathbf{Q}(\tau') = \mathbf{Q}(\tau'') \); and the values of \( \mathbf{X}(\tau), \mathbf{X}(\tau') \) and \( \mathbf{X}(\tau'') \) obey \( F \) and \( Inv^\alpha \) for all automata. If \( \tau'' \) goes to infinity, then a \( ) \) is used instead of \( ] \), and the conditions must hold for all \( \tau' \succ \tau \). A discrete step is the pair

\[ (\tau, \mathbf{X}(\tau), \mathbf{Q}(\tau)) \rightarrow (\tau', \mathbf{X}(\tau'), \mathbf{Q}(\tau')) \]

where \( \tau \) and \( \tau' \) correspond to an event point, i.e. \( \tau = (i, t) \) and \( \tau' = (i + 1, t) \); the values of \( \mathbf{X}(\tau) \) and \( \mathbf{X}(\tau') \) obey \( F \) and \( Inv^\alpha \) in their respective states; the evolution of \( \mathbf{X}(\tau) \) to \( \mathbf{X}(\tau') \) obeys the reset conditions and \( F \); and \( \mathbf{Q}(\tau) \) evolves to \( \mathbf{Q}(\tau') \) only if there are such transitions in the automata and if all guards allow it. Let \( \chi(\tau) \) denote \( (\tau, \mathbf{X}(\tau), \mathbf{Q}(\tau)) \). An execution, or trajectory, of a hybrid system is the ordered set of continuous trajectories

\[ \xi = \{ I_i \}^n_{i=1} \text{ such that} \]

\begin{itemize}
  \item \( I_i = [\chi(i, t^i_i) \rightarrow \chi(i, t^i_i)] \) is a continuous trajectory for \( i < n \) and if \( n < \infty \) then \( I_n = [\chi(n, t^*_n) \rightarrow \chi(n, t^*_n)] \) or \( I_n = [\chi(n, t^*_n) \rightarrow \chi(n, \infty)] \);
  \item \( \chi(i, t^*_i) \rightarrow \chi(i + 1, t^*_{i+1}) \) is a discrete step for all \( i \in \hat{T} \).
\end{itemize}

In analogy to the timeline definitions, it will be more convenient to think of a trajectory as \( \xi = \{ \chi : \chi \in [\chi(\tau) \rightarrow \chi(\tau'')] \in \hat{\xi} \} \). Let \( \Xi_N \) be the set of all possible trajectories, as allowed by a model \( N \).

An optimal control problem usually has a time horizon \( (N, T) \) of interest or at least a maximum \( (N^{\text{max}}', T^{\text{max}}') \). All problems will require either \( t^*_N = T \) or \( t^*_N \leq T^{\text{max}} \). The number of intervals will always be fixed by \( n = N \), which effectively handles the \( n \leq N^{\text{max}} \) case as discussed shortly. Together, these conditions define an end to a trajectory. Though this bounds trajectories in time, \( \xi \) still contains an infinite number of points because of the continuous evolution and thus is not computable. We postpone resolution of this issue to a later section. One of the benefits of equivalent modeling classes is that operations can be performed in the preferred format. Making a trajectory finite will become a problem of quantifier elimination once the model is transformed into an equation oriented style.

Unfortunately, the dummy transitions introduce a redundancy in the set of trajectories because, at some event point, all transitions could be dummy, meaning
the system has not actually changed. Also, this could occur at a continuum of time values. Thus, there are an infinite number of trajectories which are effectively equivalent. Avraam et al. (1998) recognized this problem and proposed a solution which we accommodate to this system. A *dummy event point* is one at which all automata make dummy transitions. Require all dummy event points to occur at the end of a trajectory, i.e. if \( i \in T \) is a dummy event point, then \( j \) is a dummy event point for all \( j > i \). Also, the interval length after each dummy event point should be zero, i.e. if \( i \in T \) is a dummy event point, then \( \Delta t_{i+1} = 0 \). Occurrence of a dummy event point means the effective number of intervals is less than \( n \). Thus fixing \( n \) to a constant \( N \) is only setting an upper bound on the number of intervals, obviating the need to consider \( n \leq N^{\text{max}} \). Forcing dummy event points to occur at the end selects a single trajectory amongst an infinite number of equivalent ones. This requirement is now added to the definition of a trajectory, so \( \Xi_N \) includes only trajectories with dummy event points pushed to the end.

The optimization problem (PA) requires some additional constraints for different problems. At the least, the timeline must be bounded. Periodicity constraints can be required if solving for a limit cycle, and initial and final material levels might be specified for a fluid version of the minimize makespan problem. (PA) is not solvable using existing methods because the model is presented in the newly invented framework \( N \). Transforming this optimization problem into a mixed-integer program is the subject of the latter half of this paper. But first, we present a modeling example in the new framework. At the end, we will compare this formulation against a MIP formulation.

5 Example: Modeling a Supply Network in \( N \)

The following is a model of a small supply network in the \( N \) framework. A textual input format could be provided and that is often perceived as more formal, but graphical representations can be equally formal, and we use them when appropriate. In any case, interface design is not the subject of this work. The relevant issue is the semantics being declared with the chosen syntactic/graphical input format.

\[
P = \{ A, B, C, D, X, Y, Z \}
\]

\[
t_N^e = T
\]
Figure 6: Flow network.

\[ C_u(\tau) \geq 10.0 \]

\[ \bar{m}_{p23} \text{ free } \forall p \in \{A, B, C, D\} \]

\[ C_u(\tau') = 0 \]

\[ C_u(\tau) \geq 10.0 \]

\[ C_u(\tau) \geq 75.0 \]

\[ C_u(\tau) \geq 75.0 \]

\[ m_{p34} = -M_{p3}(\tau) \forall p \in P \]

\[ C_u(\tau') = 0 \]

\[ C_u(\tau') = 0 \]

\[ C_u(\tau') = 0 \]

\[ C_u(\tau) \geq 10.0 \]

Figure 7: Shipping process \( A^u \).

\[ M_{pv}(\tau) \leq M_{pv}^{\text{max}} \quad \forall p \in P, v \in \{v_3, v_4, v_5\}, \tau \in T \quad (12a) \]

\[ \dot{C}_\alpha(\tau) = c^\alpha(Q^\alpha(i)) \quad \forall \alpha \in Q, \tau \in T \quad (12b) \]

\[ C_\alpha(\tau') = C(\tau) + \dot{c}^\alpha(i) \quad \forall \alpha \in Q, i \in \hat{T} \quad (12c) \]

The network of Figure 6 shows a complete version of the introductory example. The inventory node is replenished by an in-house supplier \( v_1 \), which sends material

\[ A + B \rightarrow X \]

\[ C \rightarrow Y \]

\[ A + 2D \rightarrow Z \]

\[ v_5 \quad \alpha \quad b \]

\[ A' \quad B \]

\[ A' \quad C \]

\[ A' \quad D \]

Table 2: Reactions \( R \), and \( \phi \).
A at a constant rate. A vendor $v_2$ supplies the other raw materials, $B$, $C$, and $D$. It can also supply $A$ if the in-house source cannot meet the demand.

Figure 7 describes the shipping process $A^u$. Node $v_3$ is a truck traveling between $v_1$ and $v_4$. The stopwatch $C_u$ enforces minimum commuting, loading, and unloading times. The amount of each material shipped is an optimization decision because $\hat{m}_{p23}$ is left free in the reset to en route.

The manufacturing process $A^r$ of Figure 8 defines a resource constrained process which must produce three materials. It governs the rates of the reactions defined in Table 2. For example, when $\text{rxnX}$ is started the reset allows an unspecified amount of feed $A$ to be input into the reactor. The same number of moles of feed $B$ are also input, the $k$ accounts for relative molecular weights. Ambient temperature is taken as an initial condition. The reaction runs at rate $6.5$ and product $X$ is removed as it is generated. When the reaction stops, unconsumed raw materials are discarded to $v_7$, and the clock is reset.

Mass conservation equations and nonnegativity of mass are implicitly by declaring the set of materials in equation (10). Finally, equations (11)–(12) define the overall system constraints $F$.

In the next section, we describe a method for automatically generating a mixed-integer program given an $N$ model of a system. We will apply the general procedure to the above example. The resulting program will be a demonstration of the transformation procedure and will also exhibit how formulations in the new formalism are simpler.
6 Model Transformations

Two options exist for optimizing a system defined in the \( \mathcal{N} \) framework: define new algorithms directly on the new framework, or transform this model into existing an format and then use existing algorithms. Of course, transformations can be viewed as the first step of an algorithm on the new model. The main distinction between the two options is in the nature of the algorithms being defined. In the first type, an algorithm would attempt to directly search on the real and finite domain spaces to find the optimal values of \( \mathbf{V} \). In the second case, the algorithm is performing symbolic operations. We choose the latter option and convert our models to MIP format because effective solution methods already exist for models written this way.

Three types of model transformations are considered. The first does not change the functional form of the model, constraints change but variables do not. If \( x \) and \( y \) are binaries, \( xy = 1 \) can be replaced by \( (x = 1) \land (y = 1) \). This amounts to employing a symbolic algebra of continuous and discrete systems. The second transformation is a reduction. The continuous time formulation makes the search space infinite. The infinite set \( \mathcal{T} \) will be replaced by a finite set \( \mathcal{T}' \subset \mathcal{T} \) in such a way that a solution to the model for all \( \tau' \in \mathcal{T}' \) fully determines the values at the remaining times.

In the third type of transformation, variables of one type are replaced with those of another type. If \( y_1, y_2 \) are Boolean and \( z_1, z_2 \) are binary, then \( y_1 \lor y_2 \) can be transformed into \( z_1 + z_2 = 1 \). The domain has been changed from \( \mathbb{R}^2 \) to \{0, 1\}\(^2\). The feasible space of the original model is \( f_1 = \{ (\text{true}, \text{false}), (\text{false}, \text{true}), (\text{true}, \text{true}) \} \) and that of the second is \( f_2 = \{ (1, 0), (0, 1), (1, 1) \} \). The two models are considered semantically equivalent because there exists a one-to-one mapping between their feasible spaces. We can associate the solution \( (1, 0) \) in the new space to the solution \( (\text{true}, \text{false}) \) of the original.

The transformations will proceed in roughly the reverse order in which models were constructed, from right to left of Figure 4. The goal is to transform \( \mathcal{N} \) into a set of MIP constraints, but transformations are discussed only to the intermediate disjunctive program (DP) because Raman and Grossmann (1994) have presented the final conversion from DP to MIP. Primary issues are generating mass conservation statements, restating automata dynamics as disjunctions, converting finite domain logic into Boolean logic, removing variable arguments, and eliminating quantifiers over infinite sets.

Strict measures of reduction in human effort are difficult to define, but some reasonable metrics can be agreed on. A smaller model is probably easier to for-
mulate, and a corollary is that only variables one actually envisions in the real
system should have to be declared. Logical constructs are closer to the concep-
tual description and presumed to be more comprehensible. Adding a process
should not require any changes to the existing model, i.e. modularity should be
supported. Discretizing time should not require redeclaring every single variable
and reformulating constraints. Finally, the model should be amenable to different
algorithms.

The following transformations serve two purposes. They provide a systematic
procedure for generating a MIP which can be sent to existing commercial software.
They also demonstrate that each of the above metrics is degraded as we move from
higher-level models such as $\mathcal{N}$ to lower-level languages such as MIP.

### 6.1 Transforming $\mathcal{N}$ to $\mathcal{M}$

Recall $\mathcal{N} = (\mathcal{M}, G, P, R)$. The transformation of $\mathcal{N}$ begins by leaving the sub-
component $\mathcal{M}$ alone and replacing $G$, $P$, and $R$ with statements added to $\mathcal{M}$. Indexing sets with the same names will remain, but that is only a syntactic con-
venience. The special meanings of these components will be removed.

The network $G$ defines where flows might occur and where they never occur. The constraints

$$
m_{puv}(q) = 0 \quad p \in P, (u, v) \notin A, q \in Q
$$

$$
\hat{m}_{puv}(i) = 0 \quad p \in P, (u, v) \notin A, i \in \hat{T}
$$

are added to $F$. Note that set $A$ is still being used and it is defined in terms of $V$, so the entire graph $G$ must still be defined to write the MIP constraints. In $\mathcal{N}$, only $G$ must be defined and the above constraints need not be declared. This is accomplished by making $V$ and $A$ have special meaning in $\mathcal{N}$—they define a flow network—while in the MIP model, they are just two indexing sets.

Material conversions $R$ are a method for efficiently declaring several related rates. For each conversion $\zeta \in R$ with $\phi(\zeta) = (v, \alpha, b)$, set $g_{\zeta pv}(q) = -r(q) \cdot \zeta_p/\zeta_b \cdot MW_p$, where $MW_p$ is the molecular weight of $p$, and similarly set $\hat{g}_{\zeta pv}(i) = -\hat{r}(i) \cdot \zeta_p/\zeta_b \cdot MW_p$. Again, the MIP constraints require all the definitions of $R$ and $\phi$ but also require that the generation rates be correctly defined by the user. Furthermore, $R$ and $\phi$ support modularity. Each $\zeta$ is a self-contained construct and $\phi$ gives a tuple with all the information required for each conversion. New conversions can be added easily. Vector notation requires inverting the definitions, causing them to be distributed amongst several parameters. One would have to
define a set of names \( \sigma \) for the conversions and then define \( \varsigma_{\sigma}, \nu_{\sigma}, \alpha_{\sigma}, \) and \( b_{\sigma} \), and each of these would have to be changed when adding a conversion. Even with these vector definitions, a richer language than provided by MIP is required because they lead to variable indices in the above constraints.

Finally, mass conservation can be guaranteed automatically. For continuous evolution, equation (2) can be generated for mass variables as

\[
\dot{M}_{\text{pv}}(\tau) = \sum_{\alpha \in \mathbf{Q} \cap \mathbf{V}} m_{\text{pvu}}^{\alpha}(Q^{\alpha}(i)) - \sum_{\alpha \in \mathbf{Q} \cap \mathbf{V}} m_{\text{pvu}}^{\alpha}(Q^{\alpha}(i)) \\
+ \sum_{\alpha \in \mathbf{Q} \cap \mathbf{R}} g_{\text{pv}}^{\alpha}(Q^{\alpha}(i)) + \dot{k} \quad \forall \tau \in \mathbf{T}, v \in \mathbf{V}, p \in \mathbf{P} \quad (14)
\]

where the constant \( k \) accounts for rates independent of any automaton. Mass conservation during discrete evolution is given by generating equation (3) as

\[
M_{\text{pv}}(\tau') = M_{\text{pv}}(\tau) + \sum_{\alpha \in \mathbf{Q} \cap \mathbf{V}} \hat{m}_{\text{pvu}}^{\alpha}(i) - \sum_{\alpha \in \mathbf{Q} \cap \mathbf{V}} \hat{m}_{\text{pvu}}^{\alpha}(i) \\
+ \sum_{\alpha \in \mathbf{Q} \cap \mathbf{R}} \hat{g}_{\text{pv}}^{\alpha}(i) \quad \forall i \in \mathbf{\hat{T}}, v \in \mathbf{V}, p \in \mathbf{P} \quad (15)
\]

In summary, elements \( \mathbf{G}, \mathbf{P}, \) and \( \mathbf{R} \) of \( \mathbf{\mathbf{N}} \) have been replaced with equivalent constraints added to element \( \mathbf{F} \) of \( \mathbf{M} \). The constraints added to \( \mathbf{F} \) are more laborious to declare, less intuitive, and error-prone. They also require all the definitions of \( \mathbf{G}, \mathbf{P}, \) and \( \mathbf{R} \) anyway. Thus, \( \mathbf{N} \) provides parts of the model, such as mass conservation, for free.

### 6.2 Transforming \( \mathbf{A} \) to Finite Domain Logic

The component automata \( \mathbf{Q} \) are the most involved constructs of model \( \mathbf{M} \). Their transformation to finite domain logic requires two steps each, and then the logic constructs can be treated together with \( \mathbf{F} \). Recall the invariant constraint for each automaton is enforced as

\[
\text{Inv}(Q(i)) \quad \forall (i,t) \in \mathbf{T}, \quad (16a)
\]

which says there is some logical relation associated with each mode, and we apply the condition for whichever mode the system is currently in. Equivalently, consider each possible value of \( Q(i) \) separately. If \( Q(i) = q \), then \( \text{Inv}(q) \) must hold,

\[
Q(i) = q \implies \text{Inv}(q) \quad \forall q \in \mathbf{Q}, \forall (i,t) \in \mathbf{T}, \quad (16b)
\]
which removes the variable argument at the expense of introducing a quantifier. The universal quantifier is a shorthand for declaring several constraints, all of which must hold; the above constraint is equivalently
\[ \bigwedge_{q \in Q} [Q(i) = q \implies \text{Inv}(q)] \quad \forall (i, t) \in T. \quad (16c) \]

For a given \( i \), consider \( Q(i) \) equal to a particular value \( q' \). Then the antecedent \( Q(i) = q \) is false for all \( q \neq q' \), making those implications true vacuously. Thus, there is just one value \( q \) for which both \( Q(i) = q \) and \( \text{Inv}(q) \) must be true. It is easy to prove that (16c) is equivalent to the disjunction over states
\[ \bigvee_{q \in Q} \left[ Q(i) = q \implies \text{Inv}(q) \right] \quad \forall (i, t) \in T. \quad (16d) \]

The term is still quantified over an infinite set and the disjunct involves a finite domain variable, but this is closer to a DP form.

A transition \((q, \gamma, \rho, q')\) means the automaton can go from state \( q \) to \( q' \) only if the guard \( \gamma \) is true and the reset \( \rho \) must also be applied,
\[ Q(i) = q \land Q(i + 1) = q' \implies \gamma \land \rho \quad \forall (q, \gamma, \rho, q') \in E, \forall i \in \hat{T}. \quad (17a) \]

This is equivalent to
\[ \bigwedge_{(q, \gamma, \rho, q') \in E} \left[ Q(i) = q \land Q(i + 1) = q' \implies \gamma \land \rho \right] \quad \forall i \in \hat{T}, \quad (17b) \]
and finally, as above, this is transformed into a disjunction over transitions
\[ \bigvee_{(q, \gamma, \rho, q') \in E} \left[ Q(i) = q \implies \gamma \land \rho \right] \quad \forall i \in \hat{T}. \quad (17c) \]

Disjunctions (16d) and (17c) can be written for all component automaton \( \alpha \in Q \), replacing \( \text{Inv}^\alpha \) and \( E^\alpha \) for each.

Model \( \mathcal{M} \)'s definition also requires removal of redundant trajectories. This can be done by requiring
\[ \forall \alpha \in Q, Q^\alpha(i) = Q^\alpha(i + 1) \implies \left[ (\forall \alpha \in Q, Q^\alpha(i + 1) = Q^\alpha(i + 2)) \land \Delta t_{i+1} = 0 \right] \quad \forall i \in \hat{T} \setminus \{N - 1\} \quad (18) \]
6.3 Eliminating Quantifiers and Variable Arguments

Consider the differential equation (2), which after integration is

\[ X(i, t) - X(i, t^*_i) = \left[ \sum_{\alpha \in Q} x^\alpha(Q^\alpha(i)) + k \right] (t - t^*_i) \quad \forall(i, t) \in T. \]  

(19)

Constraint (19) is not computable because it is quantified over an infinite set. First, divide up the quantification over each interval; \( \forall(i, t) \in T \) quantifies over the same set as \( \forall(i, t) \in [t^*_i, t^*_i], \forall i \in \mathbb{N} \). Now, for a fixed \( i \), equation (19) is linear. Thus, evaluating at only \( (i, t^*_i) \) suffices, giving

\[ X(i, t^*_i) - X(i, t^*_i) = \left[ \sum_{\alpha \in Q} x^\alpha(Q^\alpha(i)) + k \right] (t^*_i - t^*_i) \quad \forall i \in \mathbb{N}. \]  

(20)

In general, all formulas in \( L(V) \) may be quantified over all time. Such formulas arise in \( F, Inv \), and \( E \). The formulas are linear by design. Thus, the above method can be used on all statements of continuous variables. The logic statements are trivial to reduce because they are purely discrete. Indeed, we have already been replacing \( Q(i, t) \) with \( Q(i) \) throughout.

Equation (20) contains variable arguments in two forms. Those on the left are eliminated by introducing new variables \( X^e \) and \( X^s \) such that \( X^e(i) = X(i, t^*_i) \) and \( X^s(i) = X(i, t^*_i) \). The term \( x^\alpha(Q^\alpha(i)) \) on the right can be handled with a disjunction. Overall, the differential equation (2) is replaced by the two constraints

\[ X^e(i) - X^s(i) = \sum_{\alpha \in Q} w^\alpha(i) + k \Delta t_i \quad \forall i \in \mathbb{N} \]  

(21a)

\[ \bigvee_{q \in Q^\alpha} \left[ w^\alpha(i) = q \quad w^\alpha(i) = x^\alpha(q) \Delta t_i \right] \quad \forall \alpha \in Q, \forall i \in \mathbb{N}. \]  

(21b)

The disjunction correctly selects the parameter \( x^\alpha(q) \) for the state \( q \) that automaton \( A^\alpha \) is in during interval \( i \).

6.4 Finite Domain Logic to Boolean Logic

Thus far, the model consists of \( F \), disjunctions (16d) and (17c) for each automaton, and redundancy removal (18). All of these include statements of finite domain
logic $L(Q)$. Finite domain variables can be replaced by Boolean variables by introducing a Boolean variable for each value in the finite domain space, e.g. $y(on,i)$ is true only if $Q(i) = on$. In general, for each finite domain variable $Q : \mathbb{N} \rightarrow Q$, introduce Boolean variables $y : Q \times \mathbb{N} \rightarrow \mathbb{B}$, and replace $Q(i) = q$ with $y(q,i)$. This introduces $|Q|$ variables to replace each finite domain variable and expands the search space from $|Q|$ points to $2^{|Q|}$. However, if a transformation is to provide a semantically equivalent model, it can not be changing the size of the feasible space. Since the search space is larger, there must be an additional constraint. This constraint is

$$\bigvee_{q \in Q} y(q,i) \quad \forall i \in \mathbb{N} \quad (22)$$

where $\lor$ denotes exclusive or, and is required to guarantee that only one Boolean variable will be true, meaning $Q(i)$ can take only one value for each $i$.

Dummy transitions are detected by checking whether $Q(i) = Q(i+1)$. Using the above equivalence, this is $y(Q(i+1),i)$, but that makes $y$ dependent on a variable. Instead, note that $Q(i) = Q(i+1)$ is also expressed by $\bigvee_{q \in Q}[y(q,i) \land y(q,i+1)]$. For each automaton $\mathcal{A}^a$, let

$$yy^a(i) = \bigvee_{q \in Q^a} [y^a(q,i) \land y^a(q,i+1)] \quad (23)$$

mean the automaton makes a dummy transition at the $i^{th}$ event point. The $i^{th}$ event point is dummy, denoted $yyy(i)$, if all automata make dummy transitions,

$$yyy(i) = \bigwedge_{\mathcal{A}^a \in Q} yy^a(i) \quad \forall i \in \mathcal{T}. \quad (24)$$

Now equation (18) is transformed into

$$yyy(i) \implies [yyy(i+1) \land \Delta t_{i+1} = 0] \quad \forall i \in \mathcal{T}\{n-1\}, \quad (25)$$

which is equivalently the disjunction

$$[-yyy(i)] \lor \left[ \frac{yyy(i+1)}{\Delta t_{i+1} = 0} \right] \quad \forall i \in \mathcal{T}\{n-1\}. \quad (26)$$
6.5 Summary of Transformations

Eliminating infinite quantifiers and converting finite domain logic to Boolean transforms (16d) and (17c) into constraints of the type allowed in disjunctive programming. DP also allows propositional logic statements and of course algebraic constraints on reals. Thus, model $N$ has been fully converted into DP constraints. Only expressions on reals were allowed for the objective, so no conversion is required for it. In summary, optimization problem (PA) can be written as a disjunctive program,

$$\begin{align*}
\min_{X,Y} & \Omega \\
\text{s.t.} & \text{timeline definition} \\
& \text{flow restrictions between unconnected nodes (13)} \\
& \text{nonnegativity of flows and material levels} \\
& \text{mass balance over an interval, e.g. (21)} \\
& \text{mass balance at event times by discontinuity equation, e.g. (3)} \\
& \text{disjunction over states for each automaton (16d)} \\
& \text{disjunction over transitions for each automaton (17c)} \\
& \text{auxiliary equation due to change of variable type (22)} \\
& \text{dummy transition detection (23)} \\
& \text{dummy event point detection (24)} \\
& \text{redundant trajectory elimination (26)}
\end{align*}$$

A concrete example is given in the next section. The continuous variables in $X$ include all those in the original model, variables representing the event times, and auxiliary variables arising due to certain transformations, e.g. $w$ in (21). There is a Boolean variable in $Y$ for each value of each finite domain variable in $Q$ and some others that were defined for convenience, e.g. $yy$ and $yyy$.

Declaring the DP model directly is a prohibitive challenge. Numerous additional variables and constraints have been added, increasing the model size, hiding the problem structure, and making changes difficult. Furthermore, it is questionable whether the DP model would have been derived without model $N$ as a starting point.
7 Example: Generating a Disjunctive Program for a Supply Network Modeled in \( \mathcal{N} \)

The complete disjunctive programming formulation of the supply network example follows. It was derived using the transformations of the previous section.

\[
\begin{align*}
t^e_i & \geq t^s_i \quad \forall i \in \mathbb{N} \quad (27a) \\
t^s_{i+1} &= t^e_i \quad \forall i \in \hat{T} \quad (27b) \\
t^e_N &= T \quad (28)
\end{align*}
\]

\[
\begin{align*}
m^\alpha_{puv}(q) &= 0 \quad \forall \alpha \in \mathcal{Q}, p \in \mathcal{P}, (u, v) \notin \mathcal{A}, q \in \mathcal{Q}^\alpha \quad (29a) \\
m^\alpha_{puv}(i) &= 0 \quad \forall \alpha \in \mathcal{Q}, p \in \mathcal{P}, (u, v) \notin \mathcal{A}, i \in \hat{T} \quad (29b)
\end{align*}
\]

\[
\begin{align*}
m^\alpha_{puv}(q) &\geq 0 \quad \forall \alpha \in \mathcal{Q}, p \in \mathcal{P}, \forall (u, v) \in \mathcal{A}, q \in \mathcal{Q}^\alpha \quad (30a) \\
m^\alpha_{puv}(i) &\geq 0 \quad \forall \alpha \in \mathcal{Q}, p \in \mathcal{P}, \forall (u, v) \in \mathcal{A}, i \in \hat{T} \quad (30b) \\
M^s_{pv}(i), M^e_{pv}(i) &\geq 0 \quad \forall p \in \mathcal{P}, v \in \mathcal{V}, i \in \mathbb{N} \quad (30c)
\end{align*}
\]

\[
M^e_{pe}(i) - M^s_{pv}(i) = \sum_{\alpha \in \mathcal{Q}} \sum_{u \in \mathcal{V}} \sum_{q \in \mathcal{Q}^\alpha} w^1_{puv}(i) + \sum_{\alpha \in \mathcal{Q}} \sum_{v \in \mathcal{R}} \sum_{q \in \mathcal{Q}^\alpha} w^2_{puv}(i) + k \Delta t_i \quad \forall p \in \mathcal{P}, v \in \mathcal{V}, i \in \mathbb{N} \quad (31a)
\]

\[
\begin{align*}
Q^\alpha(i) &= q \\
w^1_{puv}(i) &= (m^\alpha_{puv}(q) - m^\alpha_{puv}(q)) \Delta t_i \\
w^2_{puv}(i) &= g^\alpha_{puv}(q) \Delta t_i
\end{align*}
\]

\[
M^s_{pe}(i + 1) - M^e_{pe}(i) = \sum_{\alpha \in \mathcal{Q}} \sum_{u \in \mathcal{V}} (\hat{m}^\alpha_{puv}(i) - \hat{m}^\alpha_{puv}(i)) + \sum_{\alpha \in \mathcal{Q}} \sum_{v \in \mathcal{R}} \hat{g}^\alpha_{puv}(i) \quad \forall p \in \mathcal{P}, v \in \mathcal{V}, i \in \hat{T} \quad (32)
\]
\[ g_{x,Y,p,5}(q) = -r_B^p \frac{S_p}{S_B} MW_p \quad \forall p \in P, q \in Q^r \] 
(33a)

\[ g_{x,Z,p,5}(q) = -r_C^p \frac{S_p}{S_C} MW_p \quad \forall p \in P, q \in Q^r \] 
(33b)

\[ g_{x,Z,p,5}(q) = -r_D^p \frac{S_p}{S_D} MW_p \quad \forall p \in P, q \in Q^r \] 
(33c)

\[ M^e_p(i), M^e_p(v) \leq M^{\max}_{pv} \quad \forall p \in P, v \in \{v_3, v_4, v_5\}, i \in \mathbb{N} \] 
(34)

\[
C^e_\alpha(i) - C^s_\alpha(i) = w^{3,\alpha}(i) \quad \forall \alpha \in Q, i \in \mathbb{N}
\] 
(35a)

\[
\bigvee_{q \in Q^s} \left[ \frac{w^{3,\alpha}(i) = q}{Q^s_\alpha(i)} \right] \quad \forall \alpha \in Q, i \in \mathbb{N}
\] 
(35b)

\[ C^s_\alpha(i + 1) = C^e_\alpha(i) + \epsilon^\alpha(i) \quad \forall \alpha \in Q, i \in \hat{T}
\] 
(36)

\[
\begin{bmatrix}
\begin{align*}
y^u(\text{loading}, i) \\
y^u(\text{en route}, i + 1) \\
y^u(\text{returning}, i) \\
y^u(\text{returning}, i + 1) \\
\hat{m}^u_{p34}(i) = -M^e_{p3}(i) & \forall p \in P
\end{align*}
\end{bmatrix}
\bigvee_{i \in \hat{T}} \left[ \begin{bmatrix}
\begin{align*}
y^u(\text{unloading}, i) \\
y^u(\text{returning}, i) \\
C^e_u(i) \geq 10.0 \\
C^e_u(i) \geq 10.0 \\
\hat{m}^u_{p34}(i) = 0 & \forall p \not\in \{A, B, C, D\}
\end{align*}
\end{bmatrix}
\bigvee \left[ \begin{bmatrix}
\begin{align*}
y^u(\text{loading}, i + 1) \\
y^u(\text{loading}, i) \\
C^e_u(i) \geq 75.0 \\
C^e_u(i) \geq 75.0 \\
\hat{u}(i) = 0 & \forall \hat{u} \in \hat{u}
\end{align*}
\end{bmatrix}
\right]
\bigvee \left[ \begin{bmatrix}
\begin{align*}
y^u(\text{returning}, i + 1) \\
y^u(\text{returning}, i) \\
\hat{u}(i) = 0 & \forall \hat{u} \in \hat{u}
\end{align*}
\end{bmatrix}
\right]
\bigvee \left[ \begin{bmatrix}
\begin{align*}
y^u(\text{idle}, i + 1) \\
\hat{u}(i) = 0 & \forall \hat{u} \in \hat{u}
\end{align*}
\end{bmatrix}
\right]
\bigvee \left[ \begin{bmatrix}
\begin{align*}
y^u(\text{idle}, i) \\
\hat{u}(i) = 0 & \forall \hat{u} \in \hat{u}
\end{align*}
\end{bmatrix}
\right]
\bigvee \left[ \begin{bmatrix}
\begin{align*}
y^u(\text{rxnX}, i) \\
\Theta^s(i) \leq 400 \\
\Theta^e(i) \leq 400
\end{align*}
\end{bmatrix}
\right]
\bigvee \left[ \begin{bmatrix}
\begin{align*}
y^u(\text{rxnY}, i) \\
\Theta^s(i) \leq 400 \\
\Theta^e(i) \leq 400
\end{align*}
\end{bmatrix}
\right]
\bigvee \left[ \begin{bmatrix}
\begin{align*}
y^u(\text{rxnZ}, i) \\
\Theta^s(i) \leq 600 \\
\Theta^e(i) \leq 600
\end{align*}
\end{bmatrix}
\right]
\bigvee \left[ \begin{bmatrix}
\begin{align*}
y^u(\text{setup}, i) \\
\Theta^s(i) \leq 600 \\
\Theta^e(i) \leq 600
\end{align*}
\end{bmatrix}
\right] \forall i \in \mathbb{N}
\] 
(37)
The relative sizes of the models alone indicates the increased complexity of the DP formulation. The timeline is built into the definition presented in Section 5. The important mass conservation equations (31)–(32). Table 2 is a slight convenience

$$y^r(\text{setup},i)$$
$$y^r(\text{rxnX},i+1)$$
$$C_r^e(i) \geq 2.5$$
$$\Theta_s(i+1) = 298$$
$$\hat{x}^r(i) = 0 \ \forall \hat{x}^r \in \hat{x}^r, \hat{x}^r \neq \hat{m}_{A45}^r$$

$$y^r(\text{setup},i)$$
$$y^r(\text{rxnZ},i+1)$$
$$C_r^e(i) \geq 2.0$$
$$\Theta_s(i+1) = 298$$
$$\hat{m}_{D45}^r(i) = k\hat{m}_{A45}^r(i)$$
$$\hat{x}^r(i) = 0 \ \forall \hat{x}^r \in \hat{x}^r, \hat{x}^r \neq \hat{m}_{A45}^r$$

$$y^r(\text{setup},i)$$
$$y^r(\text{rxnX},i)$$
$$C_r^e(i) \geq 1.7$$
$$\hat{m}_{A57}^r(i) = M_{A5}^e(i)$$
$$\hat{m}_{B57}^r(i) = M_{B5}^e(i)$$
$$C_r^e(i+1) = 0$$

$$y^r(\text{setup},i)$$
$$y^r(\text{rxnZ},i)$$
$$\hat{m}_{C57}^r(i) = M_{C5}^e(i)$$
$$C_r^e(i+1) = 0$$

$$\forall i \in \hat{T}$$

$$\bigvee_{q \in Q^a} y^a(q,i) \ \forall \alpha \in Q, \forall i \in N$$

$$y^a(i) = \bigvee_{q \in Q^a} [y^a(q,i) \wedge y^a(q,i+1)] \ \forall \alpha \in Q, \forall i \in \hat{T}$$

$$yy^a(i) = \bigwedge_{A^o \in Q} yy^a(i) \ \forall i \in \hat{T}$$

$$[-yyy(i)] \vee \left[ yy^a(i+1) \Delta t_{i+1} = 0 \right] \ \forall i \in \hat{T} \setminus \{n-1\}$$

Compare this disjunctive programming formulation with the \( \mathcal{N} \) formulation presented in Section 5. The relative sizes of the models alone indicates the increased complexity of the DP formulation. The timeline is built into the definition of \( \mathcal{N} \), so no declaration of it is required, but equations (27) must be added to the DP model. Equations (30) are trivial but error prone. Knowing that \( m \) and \( M \) are mass variables allows generating them automatically and also provides the important mass conservation equations (31)–(32). Table 2 is a slight convenience
over equations (33). More importantly, the reactions of $N$ are reusable, allowing libraries of reactions to be provided. Although disjunctive constraints are much simpler than their corresponding MIP constraints, they are still rather complex compared to the component automata.

Consider that a modeler was directly declaring DP constraints, and that she writes equations (29). Almost certainly, she would have first drawn herself a network of the type in Figure 6. However, this network is already a precise statement of equations (29), and the extra effort is unnecessary. The network is made a construct of the modeling language, so it serves as a constraint specification directly.

Constraint $F$ of $N$ contains several relations on real variables. Modifications to the DP are difficult because it is less general. Consider the capacity constraints (12a) in $N$ compared with (34) in the DP. It is not clear how (34) would need to be modified if the model was extended to include nonlinear dynamics. However, (12a) is unchanged. Then, given the new nonlinear dynamics, one could assess the possibility of transforming (12a) into a DP constraint. Formalizing the transformations and the conditions under which they apply allows automating them. If the transformation is not possible, the modeler must reconsider the model, no worse than the current situation. However, when the transformation is possible, it is done automatically, and the user need never change her model.

Each component automaton gives rise to two sets of disjunctions. Model $N$ provides modularity by using a single automaton for each process. The disjunctions do provide a similar feature because each disjunction embodies a distinct set of choices. However, that is not a property of disjunctions and additional definitions are required to employ the disjunctions in this way. A unique set of variables, $x$ and $\hat{x}$, are provided for each module, and the modules are linked by $F$. Additionally, the interaction of hybrid processes causes difficulties that were resolved by adding dummy transitions.

Model structure is more transparent in the automaton formulation and simplifications can be inferred more easily. For example, the stopwatch variables are not reset to zero upon entry into some states, e.g. into idle of Figure 7. However, neither the guard out of idle nor the invariant in idle make use of the stopwatch’s value. Thus, the system is unchanged if the stopwatch is reset anyway. All stopwatches can be reset on all but dummy transitions. They can be removed from the above transitions and used in a disjunction with just two terms, either the transition is dummy or not. This is important because the convex hull method for transforming disjunctions into MIP constraints increases the total number of constraints significantly, sometimes offsetting its benefit over the big-M reformu-
lation. Thus, reducing the size of disjunctions can have a significant impact on solution times. This reduction is not evident by directly observing the DP model, even after being told that it is possible.

8 Conclusions

Models formulated in the proposed framework have several advantages over MP models, the current standard for optimization problems. Models are significantly smaller, which has at least two benefits: the effort required to formulate the model is reduced and the chance of errors is minimized. More significantly, MP is a lower level modeling language and demands more knowledge from the modeler. For example, finding a linear formulation of the differential equation requires knowledge of quantifier elimination, two methods for removing variable arguments, and the convex hull or big-M method for converting disjunctions. It is not practical to expect such knowledge from end users. For experts, these are simple procedures, but we now encode their knowledge into a formal system so that they can be automated.

As higher level frameworks, $\mathcal{M}$ and $\mathcal{N}$ allow constraints to be written in a more general manner. Coupled with the conditions under which certain transformations apply, the general framework eases declaration of special purpose models. For example, the condition under which quantifications can be made finite is that all constraints must be linear. Since the condition is so well defined, the proposed modeling framework allows the modeler to declare quantifications over a real interval. Similarly, support for constructs such as reactions and networks reduces manual work.

The most significant new features are finite domain logic and component automata, which use quantification over reals, variable arguments, functional values, and tuples. These greatly narrow the gap between formal models and their conceptual representations. The automata models are virtually self explainatory, but even experts would require explanation of the corresponding DP models.

Work is currently under way to develop the software to automatically transform the new automata models into a mixed-integer program. We believe richer modeling languages coupled with automatic transformations will expand the utility of MIP methods by making them available to a larger audience. Also, the transformation technique proposed uses CP style constraints as an intermediate step. This opens the way for systematically mixing strategies from CP and MIP, and we hope to experiment with this possibility in future.
Appendix: The Set of Logic Formulae $\mathcal{L}(V)$

Several data types are being used in this work, and it is necessary to define the set of statements that are valid expressions for each of these types. The basic technique for doing so is derived from mathematical logic as discussed in Andrews (2002). Logic is also used to define programming languages, which is essentially what is being done here. Introductions are provided in Pierce (2002) and Harper (2004, unpublished).

A grammar rule defines a set recursively. For the familiar reals consider,

$$\text{rexpr ::= } k \mid X \mid \text{rexpr} + \text{rexpr} \mid k \cdot \text{rexpr}$$  \hspace{1cm} (40)

$$\text{rel ::= true \mid false \mid rexpr_1 = rexpr_2 \mid rexpr_1 \leq rexpr_2}$$

$$\mid \text{rexpr_1} \geq \text{rexpr_2} \mid \text{rel}_1 \land \text{rel}_2.$$  \hspace{1cm} (41)

Two sets are being defined. Rule (40) says that an $\text{rexpr}$ is either a real constant $k$ all by itself, a real variable $X$, two $\text{rexpr}$’s added together, or a constant multiplied by another $\text{rexpr}$. The definitions are recursive, so $\text{rexpr}$ on the right and left hand sides refer to different $\text{rexpr}$s. An $\text{rel}$ is either a propositional constant, an equation, an inequality, or a set of such relations, built up by taking conjunctions. The unary negation operator and subtraction can be added to $\text{rexpr}$s, but this is only a syntactic convenience because each can be reformulated to match an existing rule.

Similarly, valid expressions for finite domain variables are given by

$$\text{drel ::= true \mid false \mid Q = c \mid \neg \text{drel} \mid \text{drel}_1 \lor \text{drel}_2 \mid \text{drel}_1 \land \text{drel}_2},$$  \hspace{1cm} (42)

where $Q$ matches a finite domain variable from some set $Q$ and $c$ is an element of the corresponding space $Q$. Expressions on reals and finite domains can be mixed with

$$\text{rel ::= rrel \mid drel \mid rrel \land drel}.$$  \hspace{1cm} (43)

A $\text{rel}$ is either an $\text{rrel}$, a $\text{drel}$, or a mixed set of continuous and discrete relations.

The set of well formed formulas $\mathcal{L}(V)$ is defined as the set of formulas matching rule $\text{rel}$. The variables matching $X$ and $Q$ in the above grammar rules must come from $V$. Thus, $\mathcal{L}(X)$ is taken to be only those relations defined by $\text{rrel}$ and $\mathcal{L}(Q)$ only those matching $\text{drel}$.

References


