Formal Models for Control of Flexible Manufacturing Cells: Physical and System Model
Sanjay B. Joshi, Erik G. Mettala, Jeffrey S. Smith, and Richard A. Wysk

Abstract—Currently, most control implementations of flexible manufacturing cells have been developed specific to a particular facility, and no generic format or tools exist for systematic planning and creation of control. This paper presents the first phase of research in automatic generation of control software. It focuses on the development of theoretical foundations and generic issues necessary to understand and implement control. Specific formal models are developed for the physical activities, system actions, and individual machines comprising the manufacturing cell. In a subsequent paper, the formal models presented here are used to provide the context for creating context free control grammars which are used to automatically generate software for controlling flexible manufacturing cells.

I. INTRODUCTION

Currently, most control implementations of flexible manufacturing cells (FMC’s) have been developed specific to a particular facility, and no generic format or tools exist for systematic creation and planning of control software. In general, these systems are developed as “turnkey” systems by personnel other than the manufacturing engineers responsible for their operation (e.g., by integration specialists). This has resulted in systems that lack portability, often require long implementation time, have much less operational flexibility than originally anticipated, and frequently incur high implementation costs. It is clear that the need for improvements in manufacturing control activities and software exists with respect to (1) providing better algorithms for control, (2) increasing speed and reliability of software development, and (3) improving software quality.

Automatic software generation is one possible solution to the software development and maintenance problems associated with manufacturing control activities. Automatic software generation is the automatic compilation of a high level system specification into executable software. This compilation may occur in several steps (e.g., from specification language to C code and from C code to object code and from object code to executable code), but the adjective automatic implies that no human modification is required between steps. Balzer suggests that the allure of automatic code generation is based on the recognition that programming was, and still is, the bottleneck in the use of computers [2]. However, the use of automatic code generators has not been adequately addressed as pointed out by the Computer Science Technology Board (1989) which states that “the software construction process as well as the later maintenance process, is one of dealing with the details of programming, not with the nuances of the application.”

This paper presents the first phase of research in automatic generation of control software for the control of flexible manufacturing cells (FMC). The objective of this phase was to develop the theoretical foundations and identify generic issues necessary to understand and implement control of flexible manufacturing cells. The characteristics of commands and actions that control physical part movement, maintenance of system states, individual machine states and actions, and their interactions are identified and used to develop formal models of the generic control elements. These formal models are shown to be equivalent to pushdown automata (PDA) and are consequently used in the development of a novel approach to controlling these systems using context free grammars. In this approach, the individual controllers are modeled as parsers and control is exercised as a consequence of language recognition. The ability to model controllers as parsers is exploited during the automatic software generation process. This topic will be discussed in detail in a subsequent paper [12]. With the appropriate software generation tools, control systems can be developed and modified in house by the engineers responsible for their use. This will significantly reduce the cost of making future changes to the control system in response to changes in the production requirements. This paper focuses on the underlying formal models developed for FMC control. The models developed can be used to characterize control for discrete manufacturing systems like the one shown in Fig. 1.

Explicit formal models, theorems, and abstractions form a large segment of the science base in many fields. These models are considered as necessary precursors to the development
of pragmatic knowledge and tools, and provide the basic understanding of generic issues related to various classes of problems. The need for a solid science base in manufacturing has been articulated by several authors [20], [21]. One area of manufacturing that could benefit significantly from the development of formal models is the control of flexible manufacturing cells. Applications of formal models and abstractions could further the science and understanding of the control of manufacturing cells. Ostroff [19] identifies the following additional benefits of a formal framework:

- the process of formalizing informal requirements, ambiguities, omissions, and contradictions will often be discovered,
- the formal model may lead to semi-automated (or even automated) system development methods,
- the formal model can be verified for correctness by mathematical methods,
- a formally verified subsystem can be incorporated into a large system with greater confidence that it behaves as specified, and
- different designs can be compared.

A. Review of Related Research

Formal methods and grammars have been used recently in the development of various aspects of manufacturing system theory. Wu et al. [25] apply grammars to the cell formation problem, using a syntactic pattern recognition approach. Williams and Upton [24] examine the use of stochastic grammars for the measurement of flexibility. Graham and Saridis [8] describe the application of formal grammars to the control of a small robot, incorporating feedback and learning into their scheme. More importantly, their approach recognizes the usefulness of formal languages in modeling the hierarchical nature of complex control systems. The use of syntactic models in manufacturing processing and control is also presented by [24]. Tam [22] presents an approach using linguistic models to develop models for flow line production. The model is used to check production feasibility. Petri nets have also been applied to the design, testing and development of manufacturing control systems [13], [15], [16], and [26]. Bruno and Maretto [5] have developed extended Petri nets called PROT net (PROcess Translatable net) where the basic structure of the processes and synchronization can be generated automatically from the net, via translation into Ada program structures which will provide a rapid prototype of the system. Several references on the use of formal models for automatic generation of software and compilers can be found in the computer literature [1]. The use of formal models of the factory floor as a precursor for software generation has been discussed by [18]. Unfortunately, methods for systematically extending their formal models into running software have not yet evolved. One problem with their approach is that scheduling forms a large component of control, making the task more difficult.

Chaar [6] extends the model of Naylor and Volz by developing a methodology for implementing real-time control software for FMS. This methodology is based on a component-oriented rule-based specification language. Chaar uses the modeling formalism of Naylor and Volz to model the software/hardware components of the manufacturing system and the part process plans as a set of first-order logic-based rules. These rules are then coded in the specification language which can be converted to executable code. Although some possibilities for automating this conversion are mentioned, none are described. Instead, the conversion is done manually by the software engineer. However, Chaar does present an interesting method for generating a cyclic schedule for the manufacturing system. An example implementation for a prismatic machining cell is presented in [10]. A detailed description of the implementation of a material handling system as a software/hardware component is given in [9].

Biemans and Blonk [3] present a formal specification technique based on LOTOS (Language for Temporal Ordering of Specifications) for specifying CIM systems such as transport systems and work cells. Other authors [14] have developed rapid software prototyping systems that provide only for the structural specifications of cell control software, still demanding enormous development of hand crafted code to realize effective systems.

Brooks et al. [4] examine the use of formal models for implementing and verifying media access control (MAC) protocols. They support the use of formal description languages which can be automatically or semiautomatically converted into running code. Their specification language, LIFP (language for implementing fast protocols), is based on the state machine model. Ostroff presents an overview of the use of temporal logic for control of real-time discrete event systems [19].

II. THE CONTROL ENVIRONMENT

A flexible manufacturing cell (FMC) consists of two major components: hardware and controlling software. Flexibility as we refer to it, implies that the system can produce a variety of parts with little or no setup or changeover and that the routing sequence for these parts is not necessarily fixed. The hardware, which includes computer controlled machines (or NC machines), robots, storage and material handling systems, has been around for decades and problems associated with the hardware have been well studied and are reasonably well understood. The software used for FMC control can comprise several distinct components. Control software to affect part and information flow, scheduling software to affect part flow and timing, contingency planning software, and data management and collection software are all examples of these components.

Often the term control confounds these various elements into one large software element. For our purpose, control is that component responsible for managing/controlling physical actions, flow of signals and messages that initiate physical actions, and interactions with the scheduler. Control can then be viewed as the layer between scheduling and the communications interface, as shown in Fig. 2. An important aspect of viewing control in this manner is that it decouples the scheduler from the activities directly involved in executing the decisions. The “control” software in this environment provides the hooks for integration with the scheduler and other software.
This approach will allow defining the specific control activities without confounding them with other issues.

Control in a flexible manufacturing cell is often implemented in a hierarchical structure. This structure is selected for this research since it is consistent with the approach taken by the National Institute of Standards & Technology (NIST) (an approach which is widely accepted as a means of implementing control in FMC [11]). The manufacturing cell shown in Fig. 1 will be used as an example to illustrate the formal models. The cell consists of two NC machines, a material handling robot, and a load/unload port. Each of these devices represents a separately controllable entity. Fig. 3 illustrates the control hierarchy. The controllers interfaced to the machines are called workstation controllers and they interact directly with the machine controller (usually specific to a particular machine and provided by the machine tool vendor). The cell controller interacts with the workstation controllers. The formal models used for the development of the workstation and cell controllers are the subject of this paper.

To illustrate the control activities, consider a small portion of the FMC (a single robot serving two machines A and B). The robot is issued commands of the type move object from loc1 to loc2. In this command format, object is a specific part, and loc1 and loc2 are the predefined locations on the machines. Furthermore, the robot is capable of picking parts up and putting parts down at the specified locations (e.g., through a series of robot programs). Issuing such a command by the higher level controller requires checking of the logical view or state of the system, to determine if the logical and physical preconditions for successful execution of the command exist. An example of a logical precondition is that the move to a particular location be part of the required processing sequence for the part. The execution of the command may also require some physical conditions to exist. An example of a physical precondition is that the robot must be idle in order to perform the move. Successful execution of the command places the system in another state, and therefore results in a state transition (the terms state and state transition will be discussed in Section III-B). The robot controller needs to know how to process the move. For example, the move command may be executed as a sequence of pickup and putdown operations by the robot. In this case, pickup consists of grasping a part and moving it to an intermediate location, and putdown consists of moving the part from the intermediate location to the final location and releasing it. The variables (object, loc1 and loc2) will determine which robot program (or set of programs) will be executed by the robot. The physical actions of the move affect the state of the system, and the robot controller actually provides a mapping between the physical action and the system state. During the course of the pickup and putdown operations, there is a certain amount of interaction required between the robot and the machines (e.g., to ensure that the part is released by the machine only after it has been grasped by the robot). In a hierarchical, system this interaction is achieved by passing a certain sequence of messages between the robot and the machines through their mutual cell level controller. These interactions are called synchronization actions. On completion of the task there are further message interactions between the various controllers to update the system state, start the machining operations, etc. As can be seen from this simple example, there are various physical actions, system state changes, message flows, and interactions between the physical cell and the computer models that need to be accounted for in a control system to ensure error free control. The following sections describe a formal model for control in this environment.

III. TERMINOLOGY

In this paper, we shall use the idea of symbols without definition, just as “points” are not defined in geometry. Which entities are symbols will be clear from the context. Strings are composed of finite sequences of juxtaposed symbols. Letters and digits are often characterized as symbols, however, we will also include the idea of strings of characters as symbols. Thus, in our language {move, from, to, grasp, release, loc, start, stop} are all symbols. The length of a string w, denoted as |w|, is the number of symbols in the string. For example, the string move to loc has a length of 3.

A. Sets and Relations

A set is specified by a collection of objects without repetition. Designations of sets may be through the use of brackets, denoting finite sets such as [0, 1] to denote the alphabet of symbols 0 and 1, or through the use of set formers \( \{x | P(x)\} \) to denote the set of items x such that \( P(x) \) is true. \( A \cup B \), the union of A and B is \( \{x | x \in A \text{ or } x \in B\} \). \( A \cap B \), the intersection of A and B is \( \{x | x \in A \text{ and } x \in B\} \). \( A \times B \), the product of A and B, is the set of ordered pairs (a, b) such as a is in A and b is in B.

A relation is a set of pairs. The first component of each pair is selected from the domain set, and the second element of each pair is selected from the (possibly different) range set. If R is an arbitrary relation, and (a, b) is a pair in R, then aRb is said to hold, denoting truth. We are primarily concerned with relations where, if aRb and bRc, then this implies aRc, where RR → R. For example, the movement of a part from location x to location y by a material handling robot assures that the robot initially moves to location x and
grasps the object (a pick operation), followed by a movement to location $y$ and a release of the object (a put operation). Thus the relations, $x$ pick $z$ and $z$ put $y$ together imply the relation $x$ move $y$, where pick put $\rightarrow$ move.

B. Preliminary Definitions

All entities in the system are members of a universal entity set $E_u$, which contains machines, parts, programs, messages, etc. The members of the universal entity set map to elements in the real world in a natural way. $AE$ is the set of currently active entities in the system, where $AE \subset E_u$. Additionally, a one place predicate $AE(x)$ returns $x$ if $x \in AE$, and false otherwise. We elect to define first order predicates using the same names as the sets for which membership tests are desired [18]. $E_u$, $E_m$, $\ldots$ are value sets. For example, $x \in V_{part(i)}$ or equivalently $V_{part(i)}(x)$ designates that entity $x$ is a part of type $i$, and $V_{MP}(x)$ designates that entity $x$ is a manufacturing processor (MP).

A manufacturing cell is composed of $n$ logical machines. Manufacturing Devices ($MD \subset E_u$) are composed of Ports, Manufacturing Processors, Material Handlers, and possibly Automated Storage and Retrieval Units. Ports are the places where parts enter and leave the cell. As such, ports can be viewed as the physical coupling and uncoupling points between various cells and workstations. Manufacturing Processors (MP) are typically programmable NC machines, and are considered to have direct digital communication to a Cell Control Module (CCM) either through dedicated serial or network communication systems. Material Handlers (MH) are typically material handling robots which are used for loading/unloading devices within the cell. A minimal controllable cell has the elements \{Ports, MH, MP\} where $|\text{Ports}| + |\text{MH}| + |\text{MP}| = n$, with the constraints $|\text{Ports}| \geq 1$, and $|\text{MH}| \geq 1$. Note that there need not necessarily be a manufacturing processor in the cell (e.g., a robot might be used to transfer parts between two different AGV or conveyor systems). The necessary requirement for movement paths is developed later. Automated Storage and Retrieval Units (ASRS) are warehouse elements with one or more location(s) where parts may be loaded and unloaded, or transferred onto automated transport systems such as automated guided vehicles (AGV) systems, conveyors, etc. $PT \subset E_u$ is the set of parts processed by the cell. It is assumed that the part types are known during the model and software generation process. Provision for future part types can be accomplished by reserving variable labels.

$CT$ is used to describe the set of entities in contact with other entities. For example, a part program $p$ will be considered in contact with machine $y$ when the two variable predicate $CT(p, y)$ is true. Clearly, $CT \subset E_u \times E_u$. Other types of physical contacts described with $CT$ include parts in contact with machines, parts in contact with assemblies, messages in contact with controllers, etc. $PC \subset CT$ denotes the substructure relation denoting membership of parts within assemblies. For example, $SB(k, r)$ is true if $k$ is a subassembly of part $r$. Changes in the relations $AE$, $SB$, and $CT$ will occur at discrete points in time, e.g., the point at which a part is picked up or put down. We will index the points in time at which changes take place as $t_k$, $(k = 1, 2, \ldots)$. Corresponding to each $t_k$, we have a set of relations $(AE_k, SB_k, CT_k)$ which we call the state of the system a time $t_k$. We denote the set of all system states by $\Sigma$ or sometimes by $(AE, SB, CT)$, where the absence of the subscript, by slight abuse of notation, is to be interpreted as the set of all viable relations. A running system is represented as a sequence of states and state transitions. A state transition is a set of system operations which causes the system to go from one state to another (e.g., grasping a part with a robot, or processing a part on a machine tool). The objective of the manufacturing control system is to control the sequence of these state transitions in order to meet production goals. $\sigma_k = (AE_k, SB_k, CT_k) \in \Sigma$ and denotes a particular state in the sequence of state transitions $(k = 0, 1, 2, \ldots)$ representing the running system.

The $PC_k$ component of $CT_k$ is of particular interest in controlling part movement through the cell. Strictly speaking, $PC_k$ includes the identities of the parts in contact with machines. For our purposes, however, it is often sufficient to know simply whether or not there is some part in contact with a machine. We therefore define a variable $PC_k^e$, extracting this information from $PC_k$ as follows. For a cell with $n$ manufacturing devices, $PC_k^e$ is represented by a $n$ digit binary string.
$PC_k[n]PC_k[n - 1] \ldots PC_k[i] \ldots PC_k[1]$, where $PC_k[i]$ is a Boolean valued variable denoting the presence or absence of a part at machine $i$. It can be derived from the $PC_k$ relation as follows:

$$PC_k[i] = 1 \text{ if } (V_{MH}(i) \lor V_{MP}(i) \lor V_{Port}(i))$$
$$\land \exists j \text{ s.t. } V_{Port}(j) \land CT(i, j) \text{ at time } t_k$$
$$= 0 \text{ otherwise.}$$

For example, [0010] signifies that the system is made up of four manufacturing devices (called contacts) at which a part can potentially be present and that a part is currently present at the second device or contact. $|PC_k|$ is of order $2^n$ where $n$ is the number of possible contacts. For the remainder of this paper we make the following assumptions about the target environment:

1. We are dealing only with machining. There are no subassemblies and $SB$ can be dropped from the notation except where a particular tray is a member of an automated storage system (Sections IV-A-1 and IV-D-4).
2. From a control standpoint, all part types are equivalent.
   In other words, control requirements are independent of the part processing routes. Note that this is not restrictive since we are explicitly separating the control from the scheduling.
3. The set of machines, ports, and material handlers contained in $AE$ (set MD) does not change over the life of the system.

IV. A MODEL OF DISCRETE PART MANUFACTURE

The four separate views or models of the FMC used to support the development of the controllers are (each will be discussed in detail in the following sections):

1. physical model
2. system model
3. functional processing model
4. individual machine models.

The physical model provides the set of physically possible actions in the FMC, and is used primarily as the system description. The system model describes the logical conditions for successful execution of commands that cause the physical actions. For example, moving a part from point A to point B might be physically possible, but it is only logically possible if there exists a part at point A and no part at point B and there is some device available to facilitate the move. The functional processing model defines the actions performed by specific machines and devices that do not result in part movement (changes in the part contact vector $PC$), and the valid changes allowed in the system model during processing of the functional actions. The individual machine models are similar in concept to the system model in that they model the logical conditions that must exist at the machines for the physical and functional actions to take place. The system, functional, and individual machine models are not stand alone models and mappings exist between these models. Each model is defined by a set of “relations” defined with respect to “states” in the models. The relations used are:

1. $relat_p$—set of physical model relations
2. $relat_s$—set of system model relations
3. $relat_f$—set of functional relations
4. individual machine model relations:
   a) $relat_e$—robot model relations (Material Handling (MH) device)
   b) $relat_m$—machine model relations (Material Processing (MP) device)
   c) $relat_p$—port model relations (Part arrival and departure) $relat_s$—storage/ASRS model relations (Material storage).

A. Physical Model

The physical model represents the various physical actions that can occur in the FMC and the agents necessary to cause these physical actions. The physical model of the cell is represented by a node and edge list graph. Each node represents a machine in the cell, and the arcs are labeled with relations, representing physical transitions. Fig. 4 shows the graphical representation of the physical model for the FMC shown in Fig. 1. As shown in the figure, the material processing machines (NC-Lathe and NC-Mill), are represented by node MP1 and MP2, respectively. The robot is represented by node MH1 and the load/unload port is represented by node P1. The physical actions are represented by labeled arcs. For example, the putdown arc between nodes MH1 and MP1 represents the physical action of the robot (MH1) putting a part on the NC-Lathe (MP1). The figure only shows primitive physical actions (i.e., composite physical actions constructed from these primitives are not shown). The value type of a node, $St_k$ in the physical model is the machine type (e.g., MH, MP, etc.). The conditions for ensuring the validity of these physical actions are maintained by the system model (described in Section IV-B).

1) Physical Model of Part Movements: The physical model of part movements defines the primitive operations $pickup_p$ and $putdown_p$. Movement operations are generated through the composition of these primitive relations. In this
section, the primitive operations from which the movement relations are subsequently developed, are first defined.

A physical pickupp relation exists between two nodes, \( St_i \) and \( St_j \), in the physical model if the value type of \( St_i \) is a material handling robot, and the value type of \( St_j \) is either a processing machine, an automated storage device, or a port. The physical model does not account for the proper preconditions for a part movement, but only indicates that a material handling device is capable of the physical action to cause such a movement to occur. Formally, \( St_i \) pickupp \( St_j \) if \( V_{MH}(i) \land (V_{MP}(j) \lor V_{Port}(j) \lor V_{Aspect}(j)) \). This relation signifies that robot \( i \) is capable of picking a part from machine or port or ASRS port \( j \) and moving to some intermediate location. Similarly, \( St_i \) putdownp \( St_j \) if \( V_{MH}(i) \land (V_{MP}(j) \lor V_{Port}(j) \lor V_{Aspect}(j)) \). This relation signifies that robot \( i \) is capable of moving from some intermediate location and placing a part in machine or port \( j \). The forms of the pickupp and putdownp relations are identical, signifying that it is possible to execute a pickup at a given location, then it is always possible to execute a putdown at the same location.

In typical manufacturing systems, the cell controller is interested in moving parts rather than simply picking them up and putting them down. Therefore movement commands are issued to robots in the form: move \( Ob \) from \( Loc \) to \( Loc' \). Commands of this form initiate actions that are compositions of pickupp and putdownp relations. For example, execution of the move command implies the existence of a material handling robot \( k \), capable of reaching \( Loc \) and \( Loc' \), where pickupp of \( Ob \) from \( Loc \) is followed by the putdownp of \( Ob \) at \( Loc' \). Formally, \( St_i \) move\(_{gp} \) \( St_j \) if \( \exists k \in \mathbb{A} \) s.t. \( V_{MH}(k) \land St_k \) pickupp \( St_i \) \& \( St_k \) putdownp \( St_j \). The move\(_{gp} \) relation establishes the existence of the capability of a physical part movement.

2) Physical Model of Port Arrivals and Departures: Alternative materials handling systems give rise to different representations of part arrivals and departures at load/unload stations. If a conveyor is used, parts arrive and depart directly at the load/unload port. If a cart based system, such as an automated guided vehicle (AGV) system is used, then we are concerned with the representation of cart arrivals and departures from ports. The presence or absence of parts on the carts can be handled through the use of run time variables. In both cases (AGV or conveyor), part arrivals and departures from ports are designated by the pair of relations \( St_i \) arriv\(_{gp} \) \( St_j \) and \( St_i \) depart\(_{gp} \) \( St_j \), respectively. These cases are described in the following paragraphs.

Case 1: Arrival and Departure of parts on conveyors.

Subcase 1.1: Arrival of a part at a conveyor port. In this subcase, a physical arriv\(_{gp} \) relation exists between two nodes \( St_i \) and \( St_j \) if no part \( p \) is present at Port\(_i \) in \( St_i \) and some part \( q \) is present at Port\(_j \) in \( St_j \), where Port\(_i \) and Port\(_j \) are the same port. Formally, \( St_i \) arriv\(_{gp} \) \( St_j \) if \( V_{Port}(i) \land V_{Port}(j) \land (\exists q \land \neg CT(p, i) \land V_{part}(p)) \land (\forall q \land \neg CT(q, j) \land V_{part}(q)) \land (Port(i) = Port(j)) \).

Subcase 1.2: Departure of part from a conveyor port. In this subcase, a physical depart\(_{gp} \) relation exists between two nodes \( St_i \) and \( St_j \) if a part \( p \) is present at Port\(_i \) in \( St_i \) and no part \( q \) is present at Port\(_j \) in \( St_j \), where Port\(_i \) and Port\(_j \) are the same port. Formally, \( St_i \) depart\(_{gp} \) \( St_j \) if \( V_{Port}(i) \land V_{Port}(j) \land (\exists p \land \neg CT(p, i) \land V_{part}(p)) \land (\forall q \land \neg CT(q, j) \land V_{part}(q)) \land (Port(i) = Port(j)) \).

Case 2: Arrival and Departure of AGV carts.

Subcase 2.1: Arrival of a cart at an AGV port. In this subcase, a physical arriv\(_{gp} \) relation exists between two nodes \( St_i \) and \( St_j \) if no cart \( p \) is present at Port\(_i \) in \( St_i \) and some cart \( q \) is present at Port\(_j \) in \( St_j \), where Port\(_i \) and Port\(_j \) are the same port. More formally, \( St_i \) arriv\(_{gp} \) \( St_j \) if \( V_{Port}(i) \land V_{Port}(j) \land (\exists q \land \neg CT(p, i) \land V_{cart}(p)) \land (\exists q \land \neg CT(q, j) \land V_{cart}(q)) \land (Port(i) = Port(j)) \).

Subcase 2.2: Departure of a cart from an AGV port. In this subcase, a physical depart\(_{gp} \) relation exists between two nodes \( St_i \) and \( St_j \) if some cart \( p \) is present at Port\(_i \) in \( St_i \) and no cart is present at Port\(_j \) in \( St_j \), where Port\(_i \) and Port\(_j \) are the same port. Formally, \( St_i \) depart\(_{gp} \) \( St_j \) if \( V_{Port}(i) \land V_{Port}(j) \land (\forall q \land \neg CT(p, i) \land V_{cart}(p)) \land (\exists q \land \neg CT(q, j) \land V_{cart}(q)) \land (Port(i) = Port(j)) \).

3) Physical Model of Automated Storage and Retrieval: Automatic storage and retrieval (ASRS) units consist of shelves on which trays containing parts are stored. An ASRS can be commanded to retrieve a stored tray from a shelf, and position the tray for loading of parts, or for transfer onto an AGV at an Asport. The arrival of a tray at an Asport is typically associated with the movement of a tray out of the storage unit, while the departure of a tray from an Asport is typically associated with the storage of the relevant tray on an ASRS shelf.

A physical trayout\(_{gp} \) relation exists between two nodes \( St_i \) and \( St_j \) if no tray \( p \) is present at Asport\(_i \) in \( St_i \) and some tray \( q \) is present at Asport\(_j \) in \( St_j \), where tray \( p \) is a substructure of storage unit \( r \) in state \( i \) and Asport\(_i \) and Asport\(_j \) are the same port. More formally, \( St_i \) trayout\(_{gp} \) \( St_j \) if \( V_{Aspect}(i) \land V_{Aspect}(j) \land (\exists r \land \neg CT(p, i) \land V_{tray}(p)) \land (\exists r \land \neg CT(q, j) \land V_{tray}(q)) \land (Asport(i) = Asport(j)) \land \neg SB(p, r) \land \neg SB(p, r) \).

A physical trayin\(_{gp} \) relation exists between two nodes \( St_i \) and \( St_j \) if some tray \( p \) is present at Asport\(_i \) in \( St_i \) and no tray is present at Asport\(_j \) in \( St_j \), where tray \( p \) is a substructure of storage unit \( r \) in state \( i \) and Port\(_i \) and Port\(_j \) are the same port. Formally, \( St_i \) trayin\(_{gp} \) \( St_j \) if \( V_{Aspect}(i) \land V_{Aspect}(j) \land (\exists r \land \neg CT(p, i) \land V_{tray}(p)) \land (\forall q \land \neg CT(q, j) \land V_{tray}(q)) \land (Asport(i) = Asport(j)) \land \exists r V_{Aspect}(r) \land \neg SB(p, r) \land \neg SB(p, r) \).

B. System Model (Cell Controller Model—CCM)

The physical model and graph defines the arriv\(_{gp} \), depart\(_{gp} \), pickupp, putdownp, move\(_{gp} \), trayin\(_{gp} \), and trayout\(_{gp} \) relations. Each physical relation has an analog in the system model. In the system model, however, the simple relations are constrained to account for proper command preconditions, such as having a robot manipulator empty prior to executing a pickup operation, or insuring that a proper part program is resident in the controller of a particular machine before start of machining.

The relations in the system model are transitions between states represented by contact vectors. Thus, if the system
is some state \( \sigma_i \), and there is a relation \( R_k \) with another state \( \sigma_j \) such that \( \sigma_i, R_k, \sigma_j \), then there is a sequence of cell operations that will take the system from \( \sigma_i \) to \( \sigma_j \). We are primarily interested in simple one-step operations that correspond to the relations in the physical model. In order to visualize the system relations, it is useful to construct a contact state graph of the system. This graph consists of \( 2^n \) nodes (where \( n \) is the number of contacts) with each node labeled with the corresponding contact descriptor \( PC^c_k = [b^n] \) and each arc labeled with a relation from \( R^c_k \), where \( R^c_k = [b^n] \) pickup, putdown, arrive, depart, trayin, trayout. Fig. 5 shows the contact state system graph for the example cell, where specific contacts are defined in Fig. 6. Many arcs in the system graph may have the same label, denoting that the same elemental operation is required to produce a desired part movement, but that a particular part movement may exist within the context of several different state changes.

Some of the possible \( 2^n \) nodes can be omitted due to the fact that executing an action that could cause transition to a particular state may not result in a desired situation (hence must be avoided). For example, consider the transition from

\[
CT = [b b b b]
\]

Fig. 6. Example of contact relation.

[1101] to [1110] caused by the pickup of a part by the robot from the port. Now the only feasible action by the robot is a putdown, and the only available location is the port from which the part was picked. Thus, even though an action which causes a transition from state [1101] to state [1110] is possible, there will be no "real" part movement. Similar situations exist for transitions from state [1110] to state [1111]. Hence the nodes [1110] and [1111] are not included in the graph.

1) System Model of Part Movements: The pickup relation as illustrated in Fig. 7 specifies the event of a robot acquiring a part. A pickup relation exists between two states \( \sigma_i \) and \( \sigma_j \) if and only if robot \( l \) is empty in state \( \sigma_i \), and some
part $k$ is at either a processing machine, an ASRS port, or a port $m$ and if robot $l$ is grasping part $k$ in state $\sigma_j$ and the relevant processing machine or port $m$ is empty in the new state, provided that no other machine $y$ in the system changes contact state. Where the physical model $\text{pickup}_{\text{s}}$ relation represents the capability for a part movement, the system model $\text{pickup}_{\text{s}}$ relation represents a feasible part movement from a particular state of the system. Formally, $\sigma_i \text{pickup}_{\text{s}} \sigma_j$ iff $\exists k, l, m \in \text{AE}st\_\text{Vport}(k) \land \text{Vasport}(l) \land (\text{VM}(m) \lor \text{Vport}(m) \lor \text{Asport}(l) \land \forall x, y, (CT(x, y) = CT(x, y) \land \forall x, CT(x, y) \land \forall x, y, (CT(x, y))$. The $\text{putdown}_{\text{s}}$ relation, as illustrated in Fig. 8, specifies the event of a robot releasing a part. A $\text{putdown}_{\text{s}}$ relation exists between two states $\sigma_i$ and $\sigma_j$ if robot $l$ is holding some part $k$ in state $\sigma_i$, and the destination device (either a processing machine, or an Asort or a port $m$) is empty, and if robot $l$ is empty in state $\sigma_j$ and the relevant processing machine or port $m$ is holding part $k$ in the new state, provided that no other machine $y$ in the system changes contact state. Formally, the $\text{putdown}_{\text{s}}$ relation is defined by: $\sigma_i \text{putdown}_{\text{s}} \sigma_j$ iff $\exists k, l, m \in \text{AE}st\_\text{Vport}(k) \land \text{Vasort}(l) \land (\text{VM}(m) \lor \text{Vport}(m) \lor \text{Asort}(l) \land \forall x, y, (CT(x, y) = CT(x, y) \land \forall x, CT(x, y) \land \forall x, y, (CT(x, y))$.

While compositions of the primitive relations defined above are not shown on the system graph, the $\text{move}_{\text{s}}$ relation is constructed by composing a $\text{pickup}_{\text{s}}$ followed by a $\text{putdown}_{\text{s}}$ relation as shown in Fig. 9. Formally, $\sigma_i \text{move}_{\text{s}} \sigma_j$ iff $\exists k, l \in \text{AE}st\_\text{Vport}(k) \land \text{Vasort}(l) \land (\text{CT}(k, l) \land \forall x, y, (CT(x, y))$. The $\text{move}_{\text{s}}$ relation at a port: In the physical model of port arrivals and departures, two cases were presented in order to account for the logical differences encountered in conveyor and AGV processing. In conveyor systems, the model represents the status of parts at ports, while in AGV systems, the model represents the status of carts at ports. The system model must also provide for these two cases.

Case 1: Arrival and Departure of Parts on Conveyors (system case).

Subcase 1.1: Arrival of a part at a conveyor port. Formally, $\sigma_i \text{arrive}_{\text{s}} \sigma_j$ iff $\exists k, l \in \text{AE}st\_\text{Vport}(k) \land \text{Vasort}(l) \land \forall x, y, (CT(x, y)$. Subcase 1.2: Departure of a part from a conveyor port. Formally, $\sigma_i \text{depart}_{\text{s}} \sigma_j$ iff $\exists k, l \in \text{AE}st\_\text{Vport}(k) \land \text{Vasort}(l) \land (\text{CT}(k, l) \land \forall x, y, (CT(x, y)$. Case 2: Arrival and Departure of AGV carts (system case).

Subcase 2.1: Arrival of a cart at an AGV port. Formally, $\sigma_i \text{arrive}_{\text{s}} \sigma_j$ iff $\exists k, l \in \text{AE}st\_\text{Vasort}(k) \land \text{Vasort}(l) \land (\text{CT}(k, l) \land \forall x, y, (CT(x, y)$. Subcase 2.2: Departure of a cart from a conveyor port. Formally, $\sigma_i \text{depart}_{\text{s}} \sigma_j$ iff $\exists k, l \in \text{AE}st\_\text{Vasort}(k) \land \text{Vasort}(l) \land (\text{CT}(k, l) \land \forall x, y, (CT(x, y)$. 3) System Model of Automated Storage and Retrieval: In the system model of ASRS units, we are concerned only with the contact of trays with load/unload positions called $\text{Asorts}$. When a tray moves into an ASRS unit, the $\text{Asort}$ recently occupied by the tray becomes free. Similarly, when the tray moves out of the unit to support loading/unloading operations, it is positioned at an $\text{Asort}$. Due to these conditions, the ASRS system model looks remarkably similar to the Port model, as is expected. However, the conditions associated with the ASRS local model differ significantly.

A $\text{trayout}_{\text{s}}$ relation exists between two states $\sigma_i$ and $\sigma_j$ if there exists some tray $k$, not in contact with $\text{Asort}_{\text{i}}$ in state $i$, that is in contact with $\text{Asort}_{\text{i}}$ in state $j$. Formally, $\sigma_i \text{trayout}_{\text{s}} \sigma_j$ iff $\exists k, l \in \text{AE}st\_\text{Vtray}(k) \land \text{Vasort}(l) \land (\text{CT}(k, l) \land \forall x, y, (CT(x, y)$. A $\text{trayin}_{\text{s}}$ relation exists between two states $\sigma_i$ and $\sigma_j$ if there exists some tray $k$, in contact with $\text{asort}_{\text{i}}$ in state $i$, that is not in contact with the port in state $j$. Formally, $\sigma_i \text{trayin}_{\text{s}} \sigma_j$ iff $\exists k, l \in \text{AE}st\_\text{Vtray}(k) \land \text{Vasort}(l) \land (\text{CT}(k, l) \land \forall x, y, (CT(x, y)$.
### Table I
SUMMARY OF STATE SPACE AND RELATIONS

<table>
<thead>
<tr>
<th>Description</th>
<th>System (CCM)</th>
<th>Robot (MH)</th>
<th>Machine (MP)</th>
<th>Port</th>
<th>Storage (ASRS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>one state set</td>
<td>σ</td>
<td>Ω</td>
<td>Ω</td>
<td>π</td>
<td>ψ</td>
</tr>
<tr>
<td>state set relations</td>
<td>Σ</td>
<td>R</td>
<td>R</td>
<td>Π</td>
<td>Ψ</td>
</tr>
<tr>
<td>relation set</td>
<td>R&lt;sub&gt;1&lt;/sub&gt;</td>
<td>pickup&lt;sub&gt;_i&lt;/sub&gt;</td>
<td>R&lt;sub&gt;_1&lt;/sub&gt;</td>
<td>load&lt;sub&gt;_i&lt;/sub&gt;</td>
<td>R&lt;sub&gt;_1&lt;/sub&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>putdown&lt;sub&gt;_i&lt;/sub&gt;</td>
<td></td>
<td>grasp&lt;sub&gt;_i&lt;/sub&gt;</td>
<td>arrive&lt;sub&gt;_i&lt;/sub&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>move&lt;sub&gt;_i&lt;/sub&gt;</td>
<td></td>
<td>mach&lt;sub&gt;_i&lt;/sub&gt;</td>
<td>depart&lt;sub&gt;_i&lt;/sub&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>arrive&lt;sub&gt;_i&lt;/sub&gt;</td>
<td></td>
<td>unld&lt;sub&gt;_i&lt;/sub&gt;</td>
<td>hold&lt;sub&gt;_i&lt;/sub&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>depart&lt;sub&gt;_i&lt;/sub&gt;</td>
<td></td>
<td>refxt&lt;sub&gt;_i&lt;/sub&gt;</td>
<td>release&lt;sub&gt;_i&lt;/sub&gt;</td>
</tr>
</tbody>
</table>

#### C. System Model of Functional Relations

The physical and system relations previously discussed are concerned with the movement of parts in a cell. The functional relations relat<sub>_f</sub>, in contrast, are associated with actions performed by specific machines or devices which do not cause the movement of parts within the cell. For example, store<sub>_i</sub>, retrieve<sub>_i</sub>, load<sub>_i</sub>, grasp<sub>_i</sub>, release<sub>_i</sub>, machine<sub>_i</sub>, and refxt<sub>_i</sub> are the functional relations associated with the respective activities of: receiving commands to store or retrieve parts (from automated storage units), loading programs, closing fixtures, opening fixtures, executing NC programs (machining), and refixturing or repositioning tooling. relat<sub>_f</sub> functions started in state σ<sub>_i</sub> do not result in a change in the contact PC<sub>_i</sub>[0], associated with the machine processing the relat<sub>_f</sub> function, however several other contact changes might occur independent of the processing associated with relat<sub>_f</sub>. For example, when a part is machined on device i (the action associated with the mach<sub>_i</sub> relation), the system contact state variable representing the machine tool remains constant (PC<sub>_i</sub>[i] = 1). However, during the machining, other parts may have been loaded on or removed from other machines.

Each machine type in the model is defined in terms of relat<sub>_f</sub> operators that cooperate directly with associated relat<sub>_s</sub> relations in the system model. For example, when a robot puts a part on a machine tool for processing, the cell controller must synchronize the activities of the two devices so that the part is not dropped. This synchronization is handled by the cooperation of the putdown<sub>_i</sub> system relation with the putdown<sub>_i</sub> machine relation and the grasp<sub>_i</sub> functional relation. Of necessity, the notation describing the states and relations for each machine type are different. A summary of the relations used is presented in Table I. In the sections that follow, the robot, machine and port models are presented in detail.

#### D. Individual Machine Models

The individual machine models are similar in concept to the system model in that they model the logical conditions that must exist at the machines for the physical and functional actions to take place. These models map onto the system model and these mappings will be shown in the following sections.

1) Robot Model (MH Model): Similar to the system graph, each node in the robot graph is labeled with g<sub>i</sub> representing a robot state. RC<sub>_i</sub> is the n digit binary string describing parts in contact with the robot gripper which has a capacity of n parts. For example, [01] denotes a 2 gripper robot with a part in the “first” gripper. \( \emptyset = \cup_{i} g_i \) is the set of all robot states. g<sub>i</sub>R<sub>2</sub>g<sub>j</sub> where R<sub>2</sub> ∈ {pickups, putdowns} are the robot model relations and form the arc labels for the robot graph.

Fig. 10 illustrates a robot contact state graph for the single grippered robot of Fig. 1. Clearly, a single gripper can either be empty, or grasping an object, but in order to account for relative movement and positioning data associated with the gripping action, arcs are inserted into the robot graph for each corresponding relation in the system graph. In the graph, one node represents the gripper in an occupied state, and one represents an empty gripper. Each of the three arcs correspond to a pickups or putdowns between the robot R1 and either the NC Lathe, NC Mill, or Port (Fig. 1).

The pickups<sub>_i</sub> and putdowns<sub>_i</sub> relations of the robot model each map to the system model in a direct way. Fig. 11 illustrates the mapping of a pickups<sub>_i</sub> relation onto a pickups<sub>_i</sub>. In order for such a mapping to occur, some part k is grasped in the system model by some MH device l, while in the robot model, some part k is grasped by some gripper l. The pickups<sub>_i</sub> relation is defined as: g<sub>i</sub>pickups<sub>_i</sub>g<sub>j</sub> iff \( \exists k, l \in AE st V_{part}(k) \wedge V_{grip}(l) \wedge ~CT(k, l) \wedge ~ \)
CT(k, l), \exists \sigma_m, \text{pickup}_m, \sigma_n st(\text{AE}(k), \text{AE}(l)) \in \text{pickup}_m. \text{Pickup}_m \text{ relations map onto pickup}_s \text{ relations by the second existential requirement in the pickup}_s \text{ relation. While a robot model may be as simple as the two state model shown in Fig. 10, the multiple arcs between pairs of states created by the pickup}_s \text{ relations will ensure synchronization with the system model (recall that the multiple arcs represent the capability of the robot addressing multiple devices within the cell and that these multiple devices are explicitly modeled in the system graph). Similarly, } \omega_i \text{ putdown}_m \omega_j \text{ iff } \exists k, l \in \text{AE st V}_\text{part}(k) \wedge V_{\text{First}}(l) \wedge \neg CT(k, l), \wedge CT(k, l), \wedge \exists \sigma_m \text{ putdown}_m, \sigma_n st(\text{AE}(k), \text{AE}(l)) \in \text{putdown}_m. \text{ Similarly, } \omega_i \text{ release}_m \omega_j \text{ iff } \exists k, l \in \text{AE st } V_{\text{part}}(k) \wedge V_{\text{First}}(l) \wedge CT(k, l), \wedge \neg CT(k, l), \wedge \exists \sigma_m \text{ pickup}_m, \sigma_n st(\text{AE}(k), \text{AE}(l)) \in \text{putdown}_m.\]

2) Processing Machine Model (MP Model): The interaction of processing machines with the contact relations of the system model is determined by fixtures, and by functions. The states in the MP model are defined in a manner similar to the robot and system model. Each state is defined by \( \omega_i. M_i \) is the n digit binary string describing the contact between the part and the n fixtures of the machine. \( \Omega = \bigcup_i \omega_i \) is the set of all viable states. Transitions between nodes of the contact state graph of the processing machines are labeled by relations \( R_o \in \{ \text{load}_m, \text{grasp}_m, \text{release}_m, \text{mach}_m, \text{unload}_m, \text{refix}_m \} \). Fig. 12 represents a contact state graph of a processing machine with one fixture.

The \( \text{load}_m \) relation associates the interjection of an NC part program into processing machine controller. \( \omega_i \text{ load}_m \omega_j \text{ iff } \exists k, l \in \text{AE st } V_{\text{part}}(k) \wedge V_{\text{First}}(l) \wedge CT(k, l), i = j. \) The \( i = j \) condition ensures that the contact status of the machine does not change during the initiation of the program load process.

The \( \text{grasp}_m \) and \( \text{release}_m \) relations associate grasping and releasing parts on single fixtures. \( \omega_i \text{ grasp}_m \omega_j \text{ iff } \exists k, l \in \text{AE st } V_{\text{part}}(k) \wedge V_{\text{First}}(l) \wedge \neg CT(k, l), \wedge CT(k, l), \wedge \exists \sigma_m \text{ putdown}_m, \sigma_n st(\text{AE}(k), \text{AE}(l)) \in \text{putdown}_m. \text{ Similarly, } \omega_i \text{ release}_m \omega_j \text{ iff } \exists k, l \in \text{AE st } V_{\text{part}}(k) \wedge V_{\text{First}}(l) \wedge CT(k, l), \wedge \neg CT(k, l), \wedge \exists \sigma_m \text{ pickup}_m, \sigma_n st(\text{AE}(k), \text{AE}(l)) \in \text{putdown}_m. \text{ Fig. 13 illustrates the mapping of a grasp}_m \text{ relation onto a pickup}_s \text{ relation. In order for such a mapping to occur, some part must be in contact with a MH in the system model and some MP must not be in contact with a part. The grasp}_m \text{ relation then represents the machine fixture grasping the part being held by the robot.}\)

The \( \text{mach}_m \) relation, is associated with the activation of machining operations of one spindle. In effect, program \( q \) loaded on machine \( o \), which is equipped with fixture \( l \), loaded with part \( k \) receives the Start_Machining_Message, which causes the initiation of machining with spindle \( m \) in state \( j \) according to the specifications of program \( q \). Contact relations that do not change with the new state transition to active machining are not subscripted with state identifiers. Formally, \( \omega_i \text{ mach}_m \omega_j \text{ iff } \exists k, l, m, n, o, p, q \in \text{AE st } V_{\text{Start-Machining-Message}}(n) \wedge V_{\text{part}}(k) \wedge V_{\text{First}}(l) \wedge V_{\text{end}}(m) \wedge \text{VMP} (o) \wedge \text{VMachining}(p) \wedge V_{\text{mach}}(q) \wedge CT(k, l) \wedge CT(l, n) \wedge CT(n, m) \wedge \neg CT(m, q) \wedge \neg CT(m, p) \wedge CT(m, o) \wedge CT(o, n) \wedge CT(n, p) \wedge CT(p, q). \) The \( \text{unload}_m \) relation, is associated with a processing machine notification to the cell control system that machining operations are completed, and robot unloading is now permissible. \( \omega_i \text{ unload}_m \omega_j \text{ iff } \exists k, l, m, n, o, p, q \in \text{AE st } V_{\text{Machining-Finished-Message}}(n) \wedge V_{\text{part}}(k) \wedge V_{\text{First}}(l) \wedge V_{\text{unload}}(m) \wedge \text{VMP} (o) \wedge \text{VMachining}(p) \wedge V_{\text{end}}(q) \wedge CT(k, l) \wedge CT(l, n) \wedge \neg CT(m, o) \wedge \neg CT(p, o) \wedge \neg CT(n, m) \wedge CT(n, p) \wedge CT(n, q) \wedge CT(m, q) \wedge CT(m, p) \wedge CT(p, q). \) The state change that we are concerned with involves the \( CT(m, q) \) contact, which corresponds to the receipt of an Unload_Message by the cell controller, in response to the MP controller receiving notification that a part program has completed.

The \( \text{refix}_m \) relation, is associated with robot assisted refixturing operations between phases of the machining operation. \( \omega_i \text{ refix}_m \omega_j \text{ iff } \exists k, l, m, n, o, p, q \in \text{AE st } V_{\text{Refix-Required-Message}}(n) \wedge V_{\text{part}}(k) \wedge V_{\text{First}}(l) \wedge \neg CT(m, q) \wedge \neg CT(m, p) \wedge CT(m, o) \wedge CT(o, n) \wedge CT(n, m) \wedge CT(n, p) \wedge CT(p, q). \)
$V_{Refist\_Message}(m) \land V_{MP}(o) \land V_{Machining}(p) \land V_{CCM}(q) \land CT(k,l) \land CT(l,o) \land \neg CT(p,o) \land \neg CT(n,o) \land CT(n,o)$.

3) Port (Load/Unload) Model: The port control model is simply a two state model, permitting the contact of a cart (containing some number of parts) or a part with a load/unload station. The model is completed with three relations $R_3 \in \{arrive_u, depart_u, hold_u\}$. The arrive and depart relations map to the system model, and hold in the functional relation of the port model that allows holding a cart (or part) in place. Let $\pi$ represent a Port state. $\pi_{arrive_u}(x_j) \iff \exists k, l \in AE \land V_{Partition}(k) \land V_{Port}(l) \land \neg CT(k,l) \land CT(k,l) \land \exists m_{arrive_u}(AE(k), AE(l)) \in arrive_u$. Similarly, $\pi_{depart_u}(x_j) \iff \exists k, l \in AE \land V_{Partition}(k) \land V_{Port}(l) \land CT(k,l) \land \neg CT(k,l) \land \exists m_{depart_u}(AE(k), AE(l)) \in depart_u$.

For most cases it can be assumed that once the cart arrives at a location, it will depart only when a command is sent to cause departure. However, in the case where carts automatically depart after a certain time, a $hold_u$ action may be required to keep the cart in place. Fig. 14 shows the contact state graph of the port model.

4) Automated Storage and Retrieval (ASRS) Model: Automated storage and retrieval systems are designed with shelves, trays, and load/unload stations. The interaction of trays on the Asport load/unload stations form the contact relations of the system model. For an ASRS each state is defined by $\psi_i$. $\Psi = \{\psi_i\}$ is the set of all states. In a manner analogous to the $CCM$, $MH$ and $MP$ models, the contact state graph for a station ASRS consists of $n = 2^m$ nodes in the state space Transitions between states are labeled by relations $R_5$, which map ASRS transitions to $trayout_a$ or $trayout_a$ relations of the system model, or retrieve_a or store_a functional relations of the ASRS model.

A $trayout_a$ relation exists between two states $\psi_i$ and $\psi_j$ if and only there exists some tray $k$ and an automated storage port $l$ such that tray $k$ is not in contact with Asport $k$ in state $i$, and it is a substructure of some ASRS $r$ in the initial state. To complete the relation, tray $k$ is in contact with Asport $l$ in state $j$, and is not a substructure of ASRS $r$ in the final state. Additionally, tray $k$ and Asport $l$ must be active entities in a corresponding system $trayout_a$ relation. Formally, let $\psi$ represent a ASRS state. Then $\psi_{trayout_a}(x_j) \iff \exists k, l, r \in AE \land V_{Tray}(k) \land V_{Asport}(l) \land \neg CT(k,l) \land CT(k,l) \land V_{Asr}(r) \land SB(k,r) \land \neg SB(k,r) \land \exists m_{trayout_a}(AE(k), AE(l)) \in trayout_a$.

Similarly, $\psi_{trayout_a}(x_j) \iff \exists k, l, r \in AE \land V_{Tray}(k) \land V_{Asport}(l) \land \neg CT(k,l) \land CT(k,l) \land V_{Asr}(r) \land SB(k,r) \land SB(k,r) \land \exists m_{trayout_a}(AE(k), AE(l)) \in trayout_a$.

The $trayin_a$ and $trayout_a$ relations map to transitions in the system state space model. The retrieve_a and store_a relations, which we now discuss, are functional relations of automated storage units. When an ASRS controller is instructed to retrieve a specified part, the desired part may be currently on a tray in the storage unit, or on a tray that is located at an Asport. Observe that the model is only concerned about the specific time when trays are located at the load/unload ports. Otherwise, trays are considered to be in storage units. The $\psi_{retrieve_a}(x_j)$ relation exists if and only if some part $h$ is in contact with some tray $k$ in state $i$, where tray $k$ is either in contact with Asport $l$, or is located in some ASRS $n$, where ASRS $n$ is in receipt of some retrieveMsg $m$, where the retrieve message indicates the part information desired. Formally, $\psi_{retrieve_a}(x_j) \iff \exists h, k, l, m, n \in AE \land V_{Asport}(h) \land V_{Tray}(k) \land V_{Asport}(l) \land V_{RetrieveMsg}(m) \land V_{Asr}(n) \land CT(h,k) \land SB(h,n) \land CT(k,l) \land CT(m,n)$.

When the ASRS controller is instructed to store a specified part, the particular part must not currently be in the storage unit, and storage place must be available for the part. The results of the store operation is to issue a command to the ASRS to position a tray with an available storage slot at an ASRS Asport. Formally, $\psi_{store_a}(x_j) \iff \exists h, k, l, m, n, o, p \in AE \land V_{Asport}(h) \land V_{Tray}(k) \land V_{Asport}(l) \land V_{StoreMsg}(m) \land V_{Asr}(n) \land V_{Slot}(o) \land V_{Tray}(p) \land \neg CT(h,k) \land SB(h,n) \land CT(k,l) \land CT(m,n) \land \forall p \land CT(p,o) \land SB(h,k)$.

V. EQUIVALENCE OF GRAPH MODELS TO PDA MODEL

The graph models for $CCM$, $MP$, $MH$, Port and ASRS devices developed in the earlier sections can be shown to be equivalent to push down automata (PDA) (for proof refer to Appendix A). This is of fundamental importance since it establishes the fact that there must exist a set of grammars equivalent to the manufacturing cell. Given this equivalence, significant portions of the manufacturing cell controllers can be automatically generated by semantics associated with a parser generator (e.g., YACC) for the grammar. This generation will be discussed in a subsequent paper which shows how to generate the grammars and semantics associated with the models presented here [12]. Essentially a parser is generated for each controller, where the parser input comes from the communications network rather than from an input file. Control actions are performed as a consequence of parser reductions. Control actions can be physical actions such as moving a robot, loading an AGV, or machining a part, or message actions in which synchronization information is exchanged with other controllers in the system.

VI. SUMMARY AND CONCLUSIONS

In this paper we have developed a formal model of discrete parts manufacture that precisely defines the state of a man-
manufacturing system based on parts in contact with machines, and the conditions required to support movements of parts between machines. We have explicitly shown the graph model of manufacturing to be equivalent to a pushdown automata at the cell and machine level, which is of fundamental importance. Based on this equivalence, there must exist a set of grammars equivalent to the manufacturing system. These models developed herein provide the formal scientific basis for generic control of a flexible manufacturing cell. In a subsequent paper we will develop the grammar structures and search procedures to automatically generate control grammars from these formal models.

**Appendix A**

A pushdown automaton $M$ is defined by $(Q, \Sigma_p, \Gamma, \gamma, q_0, Z_0, F)$ where:
1) $Q$ is a finite set of states.
2) $\Sigma_p$ is a set of symbols called the input alphabet.
3) $\Gamma$ is a set of symbols called the stack alphabet.
4) $q_0 \in Q$ is the initial state.
5) $Z_0 \in \Gamma$ is a particular stack symbol called the start symbol.
6) $F \subseteq Q$ is a set of final states.
7) $\delta$ is the finite control function mapping $Q \times (\Sigma_p \cup \{\epsilon\}) \times \Gamma \rightarrow Q \times \Gamma^*$, where $\epsilon$ is the empty symbol.

**A. Theorem 1**

There exists a PDA equivalent for the CCM model.

**Proof:** For each $\sigma \in \Sigma$ in the CCM model, construct a state $q \in Q$ in $M$. For each arc label $\sigma_i$, place $\sigma_i$ in the graph, construct a unique symbol in $\Sigma_p$, where e.g., the three arcs labeled pickup generate the symbols pick0, pick1, pick2 corresponding to the three locations from which an object can be picked up. Add the special empty symbol $\{\epsilon\}$ to $\Sigma_p$. For each state $\sigma \in \Sigma$, construct a symbol $\gamma_i \in \Gamma$. Let $q_0 = q_0$, $Z_0 = Z_0$, and $F = \emptyset$. What remains to be shown is that $\sigma_i, $ maps to $\delta$.

Since $\sigma \in \Sigma \rightarrow q \in Q$, $\sigma_i \rightarrow Q$. Similarly, $\sigma \in \Sigma \rightarrow \gamma \in \Gamma_i$ implies $\gamma \rightarrow \Gamma_i$. Clearly $\gamma \rightarrow \Gamma_i$. By constructing $q \in Q$ and $\gamma \in \Gamma$, we can say $q \rightarrow Q \times \Gamma_i$. Observe that in the model construction process, state transitions proceed from $\sigma_i$ along $R_i$ to $\sigma_j$, where $R_i$ is defined by $\sigma_i, \gamma$. Generally then, $\Sigma_i \times R_i \rightarrow \Sigma \times \Gamma$. By definition, this is nothing but $Q \times (\Sigma_p \cup \{\epsilon\}) \times \Gamma_i \rightarrow Q \times \Gamma_i^*$, which is what was required to be shown. Therefore, an equivalent PDA can be constructed for the CCM model.

**B. Theorem 2**

There exists a PDA equivalent for the MP model.

**Proof:** For each $\omega \in \Omega$ in the MP model, construct a state $q \in Q$ in $M$. For each arc labeled $\omega_i$, place $\omega_i$ in the MP graph, construct a unique symbol $\Gamma_i$, where e.g., each arc label in Fig. 12 generates some symbol. Add the special empty symbol $\{\epsilon\}$ to $\Sigma_p$. For each state $\omega \in \Omega$, construct a symbol $\gamma \in \Gamma_i$. Let $q_0 = q_0$, $Z_0 = Z_0$, and $F = \emptyset$. What remains to be shown is that $\omega_i, \gamma \rightarrow \delta$ maps to $\delta$.

Since $\omega \in \Omega \rightarrow q \in Q$, $\omega_i \rightarrow Q$. Similarly, $\omega \in \Omega \rightarrow \gamma \in \Gamma_i$ implies $\gamma \rightarrow \Gamma_i$. Clearly $\gamma \rightarrow \Gamma_i$. By constructing $q \in Q$ and $\gamma \in \Gamma_i$, we can say $q \rightarrow Q \times \Gamma_i$. Observe that in the model construction process, state transitions proceed from $\omega_i$ along $R_i$ to $\omega_j$, where $R_i$ is defined by $\omega_i, \gamma$. Generally then, $\Sigma \times R_i \rightarrow \Sigma \times \Gamma_i$. By definition, this is nothing but $Q \times (\Sigma_p \cup \{\epsilon\}) \times \Gamma_i \rightarrow Q \times \Gamma_i^*$, which is what was required to be shown. Therefore, an equivalent PDA can be constructed for the MP model.

**C. Theorem 3**

There exist PDA equivalents for the MH, ASRS and Port models.

**Proof:** Since the details of the proof are equivalent for each case, with appropriate substitutions for $\mathcal{R}$, $\Pi$, and $\Psi$ for $\Sigma$, and $R_2$, $R_4$ and $R_5$ for $R_1$, further proofs are omitted.

**Acknowledgment**

The authors would like to express our thanks to Dr. Richard A. Volz of Texas A&M University for his contributions to this paper, in terms of his of comments and questions that helped clarify the models and identifying the notational discrepancies in the text.

**References**


Erik G. Mettala was born in Detroit, MI. He received the B.A. degree in marketing and transportation Administration from Michigan State University, the M.S. degree in computer science, and the M.S. degree in industrial and management systems engineering from the American Technological University, and the Ph.D. degree in industrial and management systems engineering from The Pennsylvania State University.

He is the Associate Dean of Engineering for Research and Professor of Computer Science and Engineering at The University of Texas at Arlington. Prior to joining UT Arlington he was the Deputy Director of the Software and Intelligent Systems Technology Office at the Advanced Research Projects Agency. In addition to his management responsibilities, he supervised several ARPA research programs including: DoD’s DEMO-II research program in autonomous robotic technology, ARPA’s Manufacturing Automation and Design Engineering (MADE) Program, which focused on the application of information technology in manufacturing; the Domain Specific Software Architecture program, enabling the composition of component-based software systems; the Real-Time Planning and Control Program; the Robotics Science program; and, he directly supervised the development of Flexible Manufacturing Technologies at Focus:HOPE in Detroit, MI. While at ARPA, he served on several national committees, including the Federal Coordinating Council for Science, Engineering and Technology—Advanced Manufacturing Technology, the Department of Defense Critical Technology Council, and the ARPA Technical Council.

He served as a member of the board of directors of the Association of Unmanned Vehicles Systems (AUVS) from 1990–1993. Prior to joining ARPA, he was the Director of Software Development for the United Kingdom’s Project Battlefield Artillery Target Engagement System (BATES). Preceding his assignment in England, he was the Chief of Systems Development and Computer Operations for the US Army’s Tactical Simulation (TACSIM) system beginning at the inception of the TACSIM program, and held numerous software development and tactical unit positions.

Jeffrey S. Smith received the B.S. degree in industrial engineering from Auburn University in 1986, and the M.S. and Ph.D. degrees in industrial engineering from the Pennsylvania State University in 1990 and 1992, respectively.

He is currently an Assistant Professor in the Industrial Engineering Department at Texas A&M University. His research interests involve the design, analysis, and control of flexible manufacturing systems, industrial robotics, flexible automation, and production systems. Currently he is working on the development of formal models of shop floor control aimed at significantly decreasing the time required to create control software for flexible manufacturing systems. He teaches courses on production control, computer aided manufacturing, discrete event simulation, and programmable automation.

Sanjay B. Joshi received the B.S. degree from the University of Bombay, Bombay, India, the M.S. degree from SUNY at Buffalo, and the Ph.D. degree in industrial engineering from Purdue University in 1987.

He is currently an Associate Professor of Industrial and Manufacturing Engineering at Penn State University. His research interests are in the areas of computer-aided design and manufacturing (CAD/CAM) with specific focus on computer-aided process planning, control of automated flexible manufacturing systems and integration of automated systems.

Dr. Joshi is the recipient of several awards, including the Presidential Young Instigator Award from the National Science Foundation in 1991, the Outstanding Young Manufacturing Engineer Award from SME in 1991, and the Outstanding Young Industrial Engineer Award from IEEE in 1993.

Richard A. Wysk is the Royce Wisenbacker Chair in Innovation and Director of the Institute for Manufacturing Systems at Texas A&M University. Prior to joining the faculty at Texas A&M, he held faculty positions at the Pennsylvania State University and Virginia Polytechnic Institute. He has also held engineering positions with General Electric and Caterpillar, Inc. He has authored or coauthored six books, and more than 80 journals papers. His research areas include computer aided process planning and flexible automation systems.

Dr. Wysk received IEEE Region III Award of Excellence (1982), the SME Outstanding Young Manufacturing Engineer (1981), Computer Aided Book of the Year Award (1992), and the SME E. Eugene Merchant Manufacturing Textbook Award (1992). He is the recipient of the IEEE David F. Baker Distinguished Research Award (1991) and is a Fellow of IEEE.